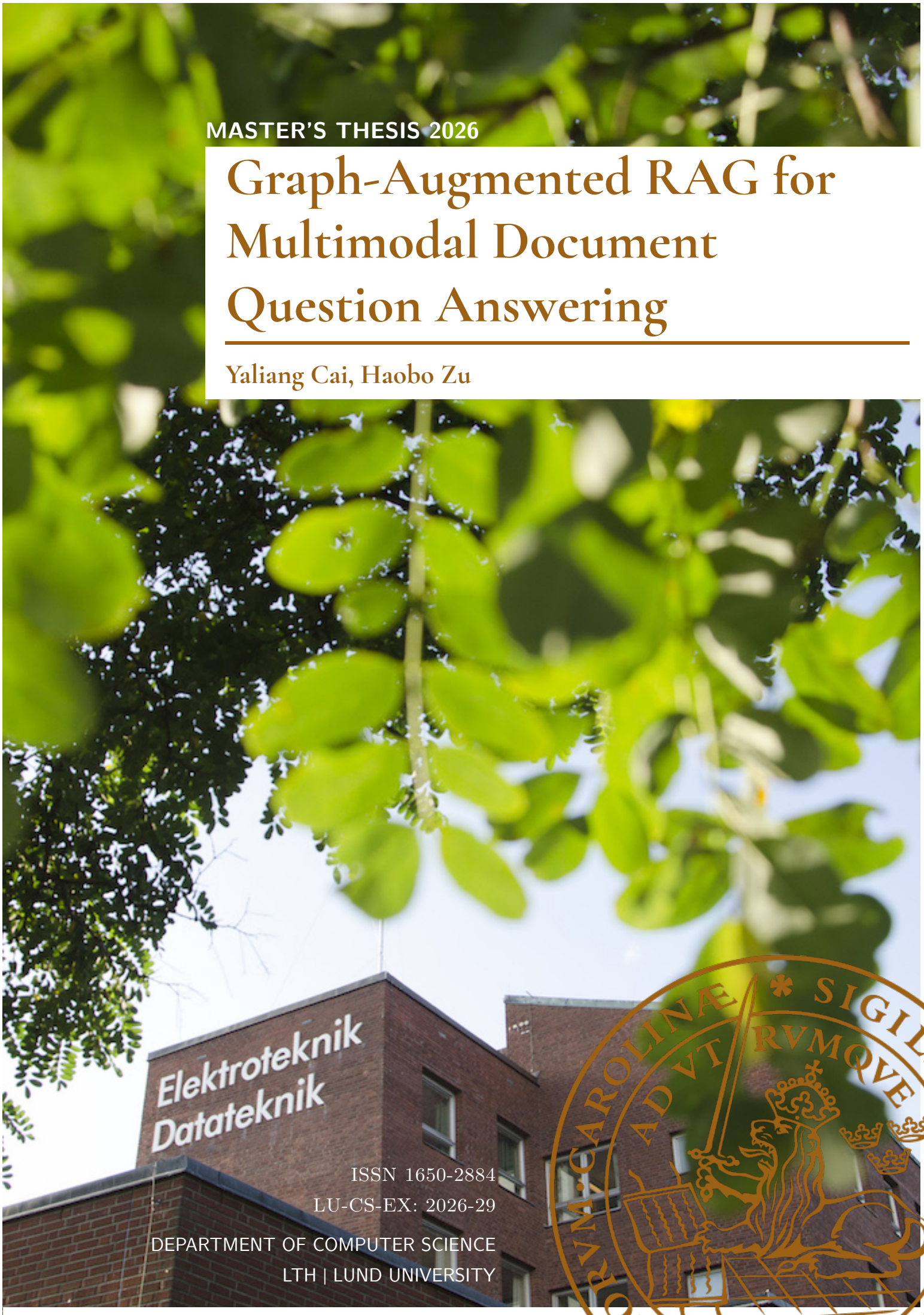


MASTER'S THESIS 2026

Graph-Augmented RAG for Multimodal Document Question Answering

Yaliang Cai, Haobo Zu

ISSN 1650-2884
LU-CS-EX: 2026-29

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2026-29

Graph-Augmented RAG for Multimodal Document Question Answering

Grafförstärkt RAG för multimodal dokumentfrågebesvarande

Yaliang Cai, Haobo Zu

Graph-Augmented RAG for Multimodal Document Question Answering

Yaliang Cai
ya8258ca-s@student.lu.se

Haobo Zu
ha2218zu-s@student.lu.se

June 4, 2026

Master's thesis work carried out at Huawei Sweden R&D.

Supervisors: Pierre Nugues, pierre.nugues@cs.lth.se
Xuhao Zhang, xuhao.zhang@huawei.com

Examiner: Xuan-Son Vu, xuan-son.vu@cs.lth.se

Abstract

Large language models require grounded, structured evidence to answer questions reliably over private, multimodal enterprise documents. Existing graph-based retrieval-augmented generation (RAG) systems address retrieval structure but rarely combine multimodal document support, genuine multi-hop graph traversal, and adaptive retrieval control in a single deployable system. We present a graph-augmented RAG system built on RAG-Anything and deployed locally to preserve data privacy. We improve knowledge graph coherence through entity resolution, enable multi-hop reasoning via Personalized PageRank (PPR) traversal combined with hybrid vector retrieval and cross-encoder reranking, and add an agentic pipeline that routes queries adaptively, detects retrieval insufficiency, and verifies answer grounding. The system is deployed as an interactive web application. Experiments on DocBench Academic, SurGE, and three multi-hop QA benchmarks show consistent improvements over the RAG-Anything baseline.

Keywords: Retrieval-augmented generation, Knowledge graph, Large language models, Multimodal document understanding, Multi-hop retrieval, Personalized PageRank

Acknowledgements

We would like to thank our supervisors, Pierre Nugues and Xuhao Zhang, for their guidance and valuable feedback throughout this thesis work.

We would also like to thank Huawei Sweden R&D for providing the opportunity and support to carry out this project.

Contents

Chapter 1
Introduction

1.1 Motivation

Large language models (LLMs) have transformed how organizations interact with large document collections. By enabling natural language queries over knowledge bases, they promise significant gains in information access and decision support. However, these models are contingent upon static corpora. They remain decoupled from private, continuously updated enterprise knowledge. This limitation circumscribes their deployment in industrial environments.

Retrieval-augmented generation (RAG) circumvents this bottleneck. It retrieves relevant evidence before generating a response, improving factual accuracy and answer traceability. As a result, RAG has become a widely adopted architecture for enterprise knowledge management.

Graph-based RAG methods reconstitute relational structure from text by constructing knowledge graphs, exploiting relational connectivity over entity networks. GraphRAG (Edge et al., 2025) pioneered this direction by using LLMs to extract entities and relations, organizing them into hierarchical community structures for global query-focused summarization. LightRAG (Guo et al., 2025b) built on this foundation, introducing a dual-level retrieval strategy that balances local entity signals with global conceptual signals, while supporting efficient incremental updates. Nevertheless, these systems remain predominantly text-centric, leading to semantic fragmentation and limited multi-hop reasoning when applied to complex industrial datasets.

Current enhancements typically follow two trajectories: LLM-guided traversal (Ma et al., 2025), which incurs prohibitive computational costs for local deployment; and hierarchical indexing (Sarthi et al., 2024), which often fails to capture fine-grained topological dependencies between technical entities.

Industrial documents, however, are rarely composed of plain text. Technical reports combine tables, figures, diagrams, and mathematical formulas. Each of these elements encodes

information that cannot be faithfully reduced to a linear text stream. Most existing RAG pipelines apply OCR or PDF-to-text conversion as a preprocessing step. This flattens document structure and discards the semantics embedded in non-textual elements. Tables lose row-column relationships; figures are reduced to captions or ignored entirely; formulas are corrupted by parsing tools. This structural flattening creates a fundamental gap between the richness of source documents and the representations available to the retrieval system.

Furthermore, although RAG-Anything (Guo et al., 2025a) extends this foundation to multimodal documents, preserving the structure of non-textual elements through layout-aware parsing and VLM-based description generation, its retrieval layer remains confined to vector matching, failing to navigate the deep logical chains connecting disparate multimodal elements. When a query requires connecting a figure to the methodology it illustrates, and from there to the quantitative result it supports, the system cannot follow this chain. Furthermore, existing architectures lack closed-loop retrieval optimization, relying on heuristic-based, single-pass strategies that cannot account for varying query complexity. This absence of adaptive retrieval mechanisms precludes the system from autonomously detecting evidence insufficiency or ensuring the alignment between retrieved context and generative outputs.

To bridge these gaps, this thesis proposes an integrated framework that extends RAG-Anything with multi-hop graph traversal, knowledge graph optimization, and an agentic retrieval pipeline, targeting document question answering (QA) on multimodal academic and enterprise corpora.

1.2 Problem Statement

We identify three categories of limitation in current graph-augmented multimodal RAG systems.

1. **Knowledge graph fragmentation.** Identity resolution in existing frameworks is based on surface-level entity names. Polysemous entities collapse into a single graph node regardless of context. Conversely, entities extracted independently from different chunks remain disconnected even when they refer to the same real-world concept. The resulting knowledge graph is fragmented: distinct concepts are merged, and equivalent concepts are isolated.

2. **Topological degradation in retrieval.** LightRAG's graph-augmented retrieval mechanism matches query keywords to entity and relation embeddings via vector similarity. This one-hop lookup ignores the relational paths encoded in the knowledge graph, discarding the multi-hop connectivity that distinguishes graph-based retrieval from flat vector search. Queries requiring reasoning across multiple relational steps—for example, tracing a visual element through its associated methodology to a quantitative result—cannot be answered from a one-hop neighborhood. Consequently, the graph's structural richness remains underutilized.

3. **Lack of retrieval intelligence and answer verification.** Single-pass retrieval systems apply a fixed retrieval strategy regardless of query complexity. When the retrieved context is insufficient, the generator hallucinates rather than requesting additional evidence.

No mechanism exists to detect retrieval failure, adapt the retrieval strategy, or verify post-hoc that a generated answer is supported by the retrieved context. In industrial settings, even a single fabricated claim can propagate into downstream decisions with high cost.

1.3 Contributions

This thesis makes the following contributions:

1. **Multi-hop graph retrieval for multimodal RAG.** We extend the one-hop retrieval of LightRAG/RAG-Anything with a Personalized PageRank (PPR) traversal layer over a chunk-augmented knowledge graph that treats document chunks as first-class nodes, producing chunk-level relevance scores directly without an entity-to-chunk projection. Combined with sparse–dense hybrid vector retrieval and reciprocal rank fusion (RRF), this yields consistent gains over the LightRAG-style baseline across 2WikiMultiHopQA, HotpotQA, and MuSiQue.

2. **Knowledge graph optimization through Entity Resolution.** We introduce a type-aware identifier scheme that prevents polysemous entities from being collapsed into a single graph node, and an embedding-based synonym-linking step that connects surface-form variants of the same entity. The combination measurably improves graph coherence (Table 6.12) and contributes to the best multi-hop QA configuration when combined with hybrid retrieval (Table 6.10).

3. **Closed-loop agentic retrieval pipeline.** We unify the retrieval components into a deterministic finite-state controller comprising an adaptive router, a multi-path retriever with weighted RRF, a sufficiency grader, a generator, and a hallucination verifier, with an explicit upper bound on language-model invocations. The agentic configuration yields the highest reported scores on all three multi-hop QA datasets (Table 6.11).

4. **Privacy-preserving local deployment with an interactive interface.** The complete system runs on local vLLM-served vision-language models (VLMs), preserving data sovereignty for proprietary documents, and is exposed to end users through a single-page web application (Section 3.5) that supports document ingestion, query answering, workspace management, and knowledge graph inspection.

We evaluate the proposed methods on three multi-hop QA benchmarks: 2WikiMultiHopQA, HotpotQA, and MuSiQue. For single-document multimodal QA, we use the Academic subset of DocBench (Zou et al., 2024). For corpus-scale retrieval recall, we use a controlled subset of SurGE (Su et al., 2026).

1.4 Thesis Outline

Chapter 2 introduces the theoretical foundations. Chapter 3 describes the system architecture and local deployment. Chapter 4 presents the knowledge graph construction pipeline and our optimization strategies. Chapter 5 details the enhanced retrieval methods. Chapter 6

describes the experimental setup and results. Chapter 7 discusses findings and limitations. Chapter 8 concludes the thesis.

1.5 Related Work

Graph-based RAG. Graph-based retrieval-augmented generation has accumulated a substantial literature in the past two years, surveyed comprehensively by Zhang et al. (2025b); we focus here on the works most directly related to our retrieval design.

GraphRAG (Edge et al., 2025) pioneered graph-structured retrieval by using LLMs to extract entities and relations from text, organizing them into hierarchical community structures for global query-focused summarization. However, it is computationally costly.

LightRAG (Guo et al., 2025b) reduces the cost of hierarchical community construction through a dual-level retrieval design (reviewed in Section 2.4.2), but its retrieval remains limited to one-hop keyword matching over entity and relation embeddings, leaving the knowledge graph underutilized, which is addressed in Chapter 5.

HippoRAG 2 (Gutiérrez et al., 2025) applies PPR over a synonym-augmented knowledge graph for associative multi-hop retrieval (Section 2.6.2); its PPR formulation and synonym-graph design directly inspire Chapters 4 and 5.

A complementary direction replaces heuristic graph algorithms with a learned model: G-Reasoner (Luo et al., 2025b) trains a query-dependent GNN foundation model over a unified graph representation for task-agnostic multi-hop reasoning. Reinforcement learning offers a further way: Graph-R1 (Luo et al., 2025a) enables end-to-end agentic graph traversal via RL; GraphRAG-R1 (Yu et al., 2026) introduces process-constrained reward signals that jointly discourage shallow retrieval and excessive reasoning steps.

Other graph-based approaches explore complementary directions: hierarchical summarization trees (Sarthi et al., 2024; Zhang et al., 2025a; Wang et al., 2025), knowledge-graph-guided traversal agents (Wang et al., 2024c), proposition-path beam search (Wang and Han, 2025), GNN-based subgraph reasoning (Mavromatis and Karypis, 2025; He et al., 2024), iterative KG–document hybrid retrieval (Ma et al., 2025), and professional domain logical inference (Liang et al., 2024); these are not directly compared in this thesis but inform the design space.

Multimodal RAG RAG-Anything (Guo et al., 2025a) extends LightRAG with layout-aware parsing via MinerU (Wang et al., 2024a) and VLM-generated descriptions for non-textual elements, so that figures, tables, and formulas participate in graph construction and vector retrieval alongside text (Section 2.4.4). This thesis builds on RAG-Anything, retaining its multimodal ingestion foundation while addressing the retrieval limitations identified above.

At the system level, recent work combines graph structure with vector search, either as a unified index, as in the Hybrid Multimodal Graph Index (Chandra et al., 2025), or in structured document pipelines that pair layout-aware parsing with hybrid retrieval over enterprise reports, such as the ESG knowledge-graph demonstrator of (CFDA and CLIP Labs, 2025). These systems emphasize index efficiency and document-structural retrieval, whereas our scheme focuses on entity-level resolution and multi-hop graph reasoning for multimodal

RAG over enterprise knowledge bases—a complementary emphasis on the same combined graph-and-vector foundation.

Entity recognition, alignment, and resolution in knowledge graphs. Constructing a coherent knowledge graph from text rests on three classical subproblems: recognizing entity mentions, aligning mentions that denote the same referent, and resolving polysemous mentions that share a surface form. Named entity recognition (NER) identifies entity mentions and assigns them to predefined semantic types; neural sequence-labeling models now dominate the task and are surveyed comprehensively by Li et al. (2022). Entity linking grounds each mention in a reference knowledge base: BLINK (Wu et al., 2020) retrieves candidates with a bi-encoder and reranks them with a cross-encoder, while GENRE (De Cao et al., 2021) generates the canonical entity name autoregressively under constrained decoding; both are reviewed by Sevgili et al. (2022). Entity alignment instead matches equivalent entities *across* graphs without a shared identifier scheme. Embedding-based methods such as MTransE (Chen et al., 2017) learn cross-graph transformations over structural embeddings, and the family is benchmarked systematically by Sun et al. (2020).

These methods assume a curated reference base, paired alignment seeds, or a clean schema. The LLM-extracted graphs of graph-based RAG present a different setting: entities, their types, and their descriptions are themselves generated outputs rather than schema fields, no external knowledge base is available under a privacy-preserving local deployment, and no labeled alignment seeds exist. Our resolution scheme (Chapter 4) therefore avoids both knowledge-base linking and trained alignment models. It reuses the type signal the extractor already produces to disambiguate polysemous mentions, and links surface-form variants through embedding similarity without merging nodes, keeping every intervention deterministic, training-free, and provenance-preserving.

Retrieval methods. Dense retrieval (Karpukhin et al., 2020) and sparse retrieval (Robertson and Zaragoza, 2009) capture complementary relevance signals. Hybrid retrieval (Luan et al., 2021) combines these signals to improve recall across query types. RRF (Cormack et al., 2009) aggregates ranked results across heterogeneous retrieval streams.

Agentic and iterative retrieval. A parallel line of work treats retrieval as an adaptive, iterative process, with the recent landscape surveyed by Singh et al. (2025). Self-RAG (Asai et al., 2024) trains models to emit retrieval tokens on demand; FLARE (Jiang et al., 2023) triggers retrieval when generation confidence drops; Adaptive-RAG (Jeong et al., 2024) routes queries to retrieval strategies of varying complexity. These systems operate over flat document collections without graph structure. Our agentic RAG pipeline (Section 5.3) extends this paradigm to graph-structured evidence, combining an adaptive query router, a sufficiency grader, a hallucination verifier, and a failure-driven escalation controller into a closed-loop system with a deterministic upper bound on LLM invocations.

Multi-hop QA benchmarks. HotpotQA (Yang et al., 2018), 2WikiMultiHopQA (Ho et al., 2020), and MuSiQue (Trivedi et al., 2022) are standard benchmarks for evaluating multi-hop reasoning over text. We use these datasets to evaluate retrieval recall in the graph traversal experiments reported in Chapter 6. DocBench (Zou et al., 2024) evaluates end-to-end LLM-based document reading across multiple evidence types and is used as the primary

Table 1.1: Comparison of related systems along five capabilities relevant to this thesis. Multi-hop: propagating relevance along knowledge graph. Multimodal: treating figures, tables, and formulas as retrievable content. Entity Resolution during graph construction. Agentic Control: adaptive retrieval with sufficiency grading and answer verification. Training-free: no task-specific model training.

Model	Multi-hop	Multimodal	Entity Resolution	Agentic Control	Training-free
Ours	✓	✓	✓	✓	✓
GraphRAG (Edge et al., 2025)					✓
LightRAG (Guo et al., 2025b)					✓
RAG-Anything (Guo et al., 2025a)		✓			✓
HippoRAG 2 (Gutiérrez et al., 2025)	✓				✓
G-Reasoner (Luo et al., 2025b)	✓				✓
Graph-R1 (Luo et al., 2025a)	✓			✓	
HMGI (Chandra et al., 2025)	✓	✓			✓
ESG-KG (CFDA and CLIP Labs, 2025)					✓
Adaptive-RAG (Jeong et al., 2024)				✓	

accuracy benchmark. SurGE (Su et al., 2026) provides a large-scale scientific literature corpus for evaluating retrieval recall at the survey level.

1.6 Author Contributions

This thesis is submitted jointly by Haobo Zu and Yaliang Cai. H. Zu and Y. Cai jointly defined the problem scope and research questions. H. Zu designed and implemented the system architecture, local deployment stack, and the multi-path retrieval and agentic components, and wrote the abstract, introduction, background, system architecture, knowledge graph, and enhanced retrieval chapters. Y. Cai refined the multimodal processing components and the entity-extraction prompts, validated and tuned the knowledge graph construction, designed and ran the evaluation, and wrote the knowledge graph construction, evaluation, discussion, and conclusion chapters.

Chapter 2
Background

This chapter introduces the technical foundations underlying this thesis. Section 2.1 sketches the Transformer-based language models that serve, in this work, as entity extractors and query answer generators. Section 2.2 discusses VLMs. Section 2.3 introduces the retrieval-augmented generation pipeline and its component techniques. Section 2.4 covers graph-based RAG, including the LightRAG and RAG-Anything systems. Section 2.6 reviews multi-hop graph-traversal algorithms, of which PPR is the one we adopt.

2.1 Transformers & LLMs

The Transformer architecture replaced recurrence with self-attention, enabling parallel computation over sequences of arbitrary length (Vaswani et al., 2017). Each layer applies two sub-layers: multi-head self-attention and a position-wise feed-forward network, each followed by residual connection and layer normalization.

2.1.1 Self-Attention

Self-attention computes dependencies between all token pairs in the input. Given input $X \in \mathbb{R}^{N \times d_{\text{model}}}$, the model projects it into query, key, and value matrices:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V \tag{2.1}$$

Attention weights are computed as scaled dot products between queries and keys, then applied to aggregate value vectors:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{2.2}$$

where $W^Q, W^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $W^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ are learnable projection matrices, N is the sequence length, d_{model} is the model hidden size, and d_k is the dimensionality of queries

and keys. The softmax is applied along the last dimension, normalizing attention weights independently for each query position. Scaling by $\sqrt{d_k}$ prevents excessively large dot products that would push the softmax into low-gradient regions, where gradients vanish and training becomes unstable.

2.1.2 Multi-Head Attention

Multi-head attention applies self-attention in parallel across h independent subspaces, allowing the model to attend to different aspects of the input simultaneously:

$$\text{MultiHead}(Q, K, V) = \text{concat}(\text{head}_1, \ldots, \text{head}_h)\, W^O, \tag{2.3}$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$. Here h is the number of attention heads, $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ are the per-head projection matrices, and $W^O \in \mathbb{R}^{hd_k \times d_{\text{model}}}$ is the output projection that combines the concatenated head outputs back into the model dimension. Each head operates in a d_k-dimensional subspace with $d_k = d_{\text{model}}/h$, so the total parameter count is comparable to that of a single full-dimensional attention layer.

2.1.3 Large Language Models

LLMs scale the decoder-only Transformer to billions of parameters through pretraining on large text corpora. Autoregressive next-token prediction is the primary training objective. Through this process, LLMs acquire broad language understanding and encode factual knowledge in their parameters.

Despite strong capabilities, LLMs have two key limitations in industrial settings. First, their knowledge is static: it reflects the training data and cannot be updated without retraining. Second, they are prone to hallucination, generating plausible but factually incorrect content. These limitations motivate retrieval-based grounding during inference.

2.2 Vision-Language Models

VLMs extend Transformer-based language models to consume images alongside text. A typical VLM comprises a vision encoder, commonly a Vision Transformer (Dosovitskiy et al., 2021), that emits a sequence of patch embeddings, together with a projection layer that maps those embeddings into the token-embedding space of a language-model backbone. The backbone then processes the concatenated sequence of visual and textual tokens through autoregression.

VLMs play two distinct roles in this thesis. At *indexing time*, non-textual document elements (figures, tables, formulas) are submitted to the VLM, which generates a natural-language description that subsequently enters the text-only retrieval and graph-construction pipeline (Chapter 3). At *query time*, the retrieved chunks, together with the original image crops, are assembled into the generation prompt, allowing the VLM to ground its answer in the raw visual content rather than in the caption or description alone.

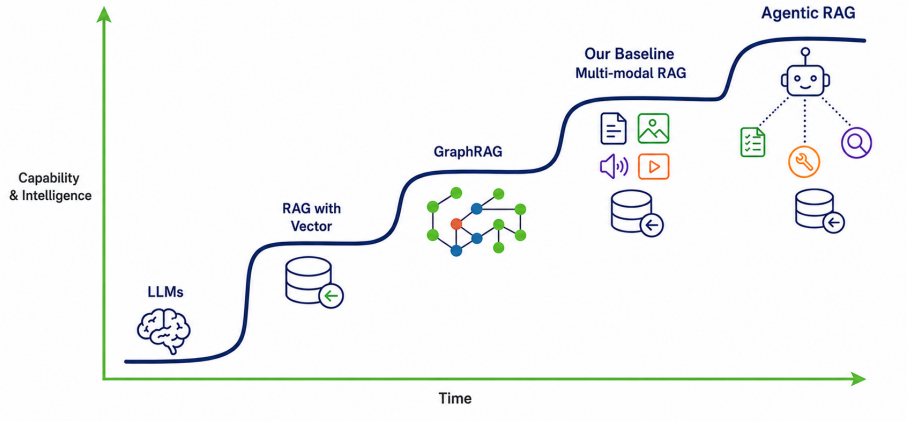


Figure 2.1: The evolution of RAG paradigms, from LLMs through vector-based and graph-based retrieval to multimodal and agentic systems. The multimodal stage corresponds to RAG-Anything (Guo et al., 2025a), the baseline this thesis builds on. Figure adapted from Neo4j (2024).

2.3 Retrieval-Augmented Generation

RAG grounds LLM responses in external evidence retrieved at query time (Lewis et al., 2020). The field has progressed through several paradigms (Figure 2.1): vector retrieval, graph-based retrieval, multimodal extension, and most recently agentic pipelines.

A RAG pipeline consists of two phases. At *indexing time*, documents are parsed and split into chunks. Each chunk is encoded into a dense vector by an embedding model and stored in a vector database (VDB). A sparse index can be constructed in parallel to support keyword-based retrieval. Chunk size is a critical design choice: chunks that are too small fragment relevant information across multiple entries, while chunks that are too large introduce noise that degrades retrieval precision. At *query time*, the system retrieves the chunks most relevant to the query, optionally reranks them, and supplies them as context to an LLM that generates the final answer.

2.3.1 Retrieval Strategies

Sparse retrieval. BM25 (Robertson and Zaragoza, 2009) ranks a document d against a query $q = (q_1, \ldots, q_n)$ by summing a saturated, length-normalized contribution from each query term:

$$\mathrm{BM25}(q,d) = \sum_{i=1}^{n} \mathrm{IDF}(q_i) \cdot \frac{f(q_i,d) \cdot (k_1+1)}{f(q_i,d) + k_1 \cdot \left(1 - b + b \cdot \dfrac{|d|}{\mathrm{avgdl}}\right)}, \tag{2.4}$$

where $f(q_i, d)$ is the frequency of term q_i in document d, $|d|$ is the length of d in tokens, avgdl is the average document length in the corpus, and $\mathrm{IDF}(q_i)$ is the inverse document frequency of q_i, which down-weights terms that appear in many documents. Two free parameters shape the response: $k_1 \in [1.2, 2.0]$ controls term-frequency saturation, so that repeated occurrences

of a term contribute progressively less, and $b \in [0, 1]$ (typically 0.75) controls how aggressively long documents are penalised relative to the corpus average. Sparse retrieval excels at exact keyword matching but fails on synonyms and paraphrases, which motivates the dense and hybrid variants below.

Dense retrieval. Dense passage retrieval (DPR) (Karpukhin et al., 2020) encodes the query q and each candidate document d into fixed-dimensional vectors $\mathbf{e}_q, \mathbf{e}_d \in \mathbb{R}^{d_{\text{emb}}}$ using a pretrained Transformer encoder, and ranks documents by cosine similarity $\cos(\mathbf{e}_q, \mathbf{e}_d) = \mathbf{e}_q^\top \mathbf{e}_d / (\|\mathbf{e}_q\| \, \|\mathbf{e}_d\|)$. The encoder is typically fine-tuned with a contrastive objective so that semantically related query–document pairs are mapped to nearby points in the embedding space. Unlike sparse retrieval, dense retrieval can match paraphrases and synonyms that share no surface form, but it may miss rare or domain-specific terms that the encoder has not seen during pretraining. Modern retrieval encoders such as BGE-M3 (Chen et al., 2024) are trained on large multilingual corpora and additionally output sparse lexical weights and multi-vector representations from a single forward pass, supporting hybrid retrieval modes within a unified model.

Hybrid retrieval. Hybrid retrieval (Luan et al., 2021) combines sparse and dense signals to leverage their complementary strengths, and is the regime adopted by our system.

Reciprocal Rank Fusion (RRF). When several retrievers return overlapping but differently ordered candidate sets with incomparable score scales, RRF (Cormack et al., 2009) fuses their rankings by summing reciprocal ranks across retrievers:

$$\text{RRF}(d) = \sum_{i=1}^{R} \frac{1}{k + r_i(d)}, \tag{2.5}$$

where R is the number of retrievers being fused, $r_i(d)$ is the rank assigned to document d by retriever i (with documents not retrieved by i either omitted or treated as having infinite rank, contributing zero), and k is a smoothing constant that dampens the influence of top-ranked documents and limits the contribution of any single retriever. We follow the standard setting $k = 60$ from Cormack et al. (2009). RRF requires no score calibration across retrievers, which makes it well-suited for fusing heterogeneous ranking signals such as sparse, dense, and graph-based retrieval.

2.3.2 Cross-Encoder Reranking

Bi-encoder retrievers prioritise recall over precision because they embed the query and the document independently. A cross-encoder reranker (Nogueira and Cho, 2019) refines the top-k candidates by jointly encoding each query–document pair through a Transformer:

$$\text{score}(q, d) = \mathbf{w}^\top h_{[\text{CLS}]}([\text{CLS}]\ q\ [\text{SEP}]\ d\ [\text{SEP}]), \tag{2.6}$$

where the query q and document d are concatenated into a single input sequence delimited by the special tokens `[CLS]` and `[SEP]`, $h_{[\text{CLS}]}(\cdot) \in \mathbb{R}^{d_{\text{model}}}$ denotes the final-layer hidden vector at the `[CLS]` position, and $\mathbf{w} \in \mathbb{R}^{d_{\text{model}}}$ is a learned scoring head that projects this vector to a

scalar relevance score. Because q and d attend to each other through every Transformer layer, the model captures fine-grained token-level interactions that bi-encoders cannot represent. The cost is quadratic in the combined sequence length, so reranking is applied only to a small candidate pool returned by an upstream retriever.

2.3.3 Generation

The retrieved chunks and the original query are assembled into a prompt and passed to the LLM. The prompt instructs the model to ground its answer in the retrieved context, which reduces but does not eliminate hallucination. A key constraint is the LLM's context window: the combined length of the query, retrieved chunks, and prompt instructions must not exceed this limit, which in practice caps the number of chunks supplied per query.

2.4 Graph-based RAG

A knowledge graph (KG) represents information as typed entities connected by directed, labeled relations: a set of triples (h, r, t) in which h and t are entity nodes and r is a relation edge. This structure supports multi-hop reasoning—queries that are answered by traversing a chain of relations across multiple entities rather than by matching a single chunk.

2.4.1 LLM-based Knowledge Graph Construction

In graph-based RAG, knowledge graphs are not curated by hand but extracted directly from document corpora by an LLM. For each chunk, the model is prompted to generate a structured list of entities, each with a name, a type, and a short natural-language description, and a list of relations, each carrying a source entity, a target entity, a relation label, and a description.

GraphRAG (Edge et al., 2025) established this paradigm and supplements it with hierarchical community structures and global query-focused summarization. GraphRAG's main drawback, however, is its high is its high computational latency and substantial token costs, driven by extensive LLM inference during both knowledge graph construction and query-time global summarization.

2.4.2 LightRAG Architecture

LightRAG (Guo et al., 2025b) reduces the query-time cost of GraphRAG by replacing community summarisation with a dual-level retrieval design. Entities and relations are stored in two parallel structures: a graph for structural traversal and a vector database for embedding-based search. Entity and relation descriptions are embedded independently, allowing semantic search over both nodes and edges; a key–value store complements these structures by caching chunk content, entity and relation metadata.

At query time, the query is first passed to an LLM that extracts two types of keywords: *high-level keywords* capturing the abstract theme of the query (what it is about), and *low-level keywords* naming specific entities or concepts (who or what it concerns). High-level keywords

are matched against the relation vector database; low-level keywords are matched against the entity vector database. The retrieved entities and relations are then expanded by fetching their neighbouring nodes and connected edges from the graph, assembling a local subgraph as structured context.

Retrieved entities and relations are not returned directly to the LLM. Each matched record carries a set of *source chunk IDs* referring to the document chunks from which it was extracted; LightRAG uses these IDs to look up the corresponding raw text from the key–value store. The final context delivered to the LLM therefore contains three layers: entity descriptions, relation descriptions, and the original chunks that grounded each extraction, which together preserve traceability from the generated answer back to its source.

2.4.3 Layout-Aware Document Parsing (MinerU)

Industrial documents combine text with tables, figures, diagrams, and mathematical formulas. Standard OCR-based pipelines extract text in reading order and discard the spatial layout and semantic structure of non-textual elements, with consequences for downstream retrieval that we discuss in Section 5.1. Layout-aware parsing treats the document as a structured object rather than a flat text stream. MinerU (Wang et al., 2024a) integrates layout analysis, formula recognition via LaTeX conversion, and table structure recognition into a single pipeline whose Markdown output preserves the spatial semantics of each document element.

2.4.4 RAG-Anything: Multimodal Extension

RAG-Anything (Guo et al., 2025a) extends LightRAG to multimodal documents by prepending the MinerU-based parsing stage of Section 2.4.3. Non-textual elements are passed to a VLM that generates a textual description of each element; these descriptions enter the standard LightRAG indexing pipeline as ordinary text chunks, so multimodal content participates in both graph construction and vector retrieval without a separate modality-specific subsystem. An alternative paradigm represented by ColPali (Faysse et al., 2024) embeds entire document pages in a visual embedding space and retrieves directly without an intermediate text representation, trading textual entity structure for higher visual fidelity. This thesis builds on RAG-Anything as its ingestion foundation and extends its retrieval layer in Chapters 4 and 5.

2.5 Serving and Storage Infrastructure

The system described in Chapter 3 depends on three pieces of mature infrastructure: a high-throughput model serving runtime, a graph database, and a vector database. We introduce each briefly before they recur in the system description.

vLLM. vLLM is an open-source serving system for large language models built around PagedAttention (Kwon et al., 2023), an attention algorithm that manages the key–value cache in fixed-size blocks rather than as one contiguous buffer per request. The block layout, borrowed from operating-system virtual memory, eliminates external fragmentation, allows the

cache of completed requests to be released at block granularity, and supports prefix sharing across requests that begin with identical token sequences. The latter property, known as *automatic prefix caching*, is directly exploited by our agentic pipeline (Section 5.3): the grader, generator, and verifier all condition on the same retrieved-chunk context, so vLLM reuses the prefill computation across these calls. vLLM exposes an OpenAI-compatible REST endpoint, which lets the same client code drive a locally hosted model and a hosted API without modification.

Neo4j. Neo4j (Robinson et al., 2015) is a native graph database that stores nodes and relationships as first-class objects rather than as rows in adjacency tables, an approach the Neo4j literature refers to as *index-free adjacency*: each node holds direct pointers to its incident edges, so traversing from a node to its neighbours is constant-time and independent of the total graph size. The query language is Cypher, a declarative graph pattern language. For this thesis the relevant capabilities are indexed lookup of entities by identifier, neighbourhood expansion, and full export of the workspace adjacency to an in-memory sparse matrix for PPR computation (Section 5.2).

Qdrant. Qdrant (Qdrant Team, 2024) is an open-source vector similarity search engine. Vectors are organized into logical *collections*, each with a fixed dimension and distance metric; approximate-nearest-neighbour (ANN) search is implemented using the Hierarchical Navigable Small World index (Malkov and Yashunin, 2020), which constructs a multi-layer proximity graph that supports sub-linear query time while approximating the exact nearest-neighbour ranking. We maintain three separate collections per workspace—chunks, entities, and relations—so that retrieval can target each level of granularity independently. Qdrant also supports payload filtering, which we use to restrict retrieval to the active workspace without partitioning the underlying storage.

2.6 Graph Traversal for Multi-Hop Retrieval

This section introduces graph terminology (Section 2.6.1) and reviews the PPR algorithm (Section 2.6.2) that we adopt in Chapter 5; two learned alternatives are surveyed (Section 2.6.3)

2.6.1 Graph Terminology

A graph $G = (V, E)$ consists of a set of nodes V and a set of edges E. In a knowledge graph, nodes represent entities and edges represent typed relations. The *neighbours* of a node v are all nodes connected to v by a single edge, $\mathcal{N}(v) = \{u \mid (v, u) \in E\}$; a *path* of length k is a sequence $v_0, v_1, \ldots, v_k$ in which every consecutive pair is joined by an edge; and a query that can be answered by traversing k edges from the seed entities is called a *k-hop* query.

2.6.2 PageRank

PageRank (Page et al., 1999) propagates a relevance signal from a set of seed nodes across the entire graph. Given a seed node s and a teleport probability $\alpha \in (0, 1)$, the PPR score $\pi_s(v)$

of node v satisfies

$$\pi_s(v) = (1-\alpha) \sum_{u \in \mathcal{N}(v)} \frac{\pi_s(u)}{|\mathcal{N}(u)|} + \alpha \cdot \mathbf{1}[v = s], \tag{2.7}$$

where $\mathcal{N}(u)$ denotes the neighbours of node u (defined in Section 2.6.1), $|\mathcal{N}(u)|$ is its degree, and $\mathbf{1}[v = s]$ is the indicator function, equal to 1 when $v = s$ and 0 otherwise. The recurrence has two terms: with probability $1 - \alpha$ a random walker at node u steps uniformly to one of its neighbours, contributing $\pi_s(u)/|\mathcal{N}(u)|$ of its mass to each; with probability α the walker teleports back to the seed s, which is the role of the indicator term. The PPR distribution π_s is the unique stationary distribution of this Markov chain. When the seed is replaced by a probability distribution $\mathbf{s}$ over multiple nodes, the indicator term generalises to $\alpha \cdot \mathbf{s}(v)$, which is the form used by our retrieval pipeline (Section 5.2). Nodes that are structurally close to the seed—reachable via short paths or multiple routes—accumulate higher scores, so PPR extends retrieval beyond the immediate neighborhood and propagates query relevance across multi-hop relational chains.

2.6.3 Learned and Adaptive Alternatives

Two parallel lines of work replace heuristic traversal with learned control. *Graph neural networks* (Kipf and Welling, 2017) aggregate information from local neighbourhoods through layer-wise message passing; stacking k GNN layers integrates the k-hop neighborhood of each node, and recent systems such as G-Reasoner (Luo et al., 2025b) train a query-dependent GNN as a foundation model over a unified graph representation. *Reinforcement-learning* approaches such as GraphRAG-R1 (Yu et al., 2026) and Graph-R1 (Luo et al., 2025a) train an LLM to decide when and how deeply to traverse the graph, using rewards that jointly discourage shallow retrieval and excessive reasoning. Both directions are promising replacements for the deterministic controller adopted in this thesis and are discussed as future work.

Chapter 3
System Architecture

This chapter describes the architecture of the multimodal RAG system and its local deployment configuration. The system is built on two open-source foundations: RAG-Anything for multimodal document ingestion (Guo et al., 2025a) and LightRAG for graph-based RAG framework (Guo et al., 2025b). Its central design principle is a uniform treatment of heterogeneous document modalities (text, tables, figures, and mathematical formulas) that preserves semantic structure throughout the pipeline rather than flattening every element into a linear text prior to chunking.

Text-centric RAG systems typically discard the spatial and semantic relationships encoded in document layout: tables lose their row–column associations, figures collapse to captions, and formulas degrade into unparseable symbol sequences. The pipeline described here replaces that lossy normalization with a four-stage process that maintains modality boundaries at every level of processing, exposes the resulting structure through a web service, and isolates multi-document corpora through an explicit workspace abstraction. Figure 3.1 gives an overview, and the remainder of this chapter elaborates each stage in turn.

3.1 Multimodal Parsing Pipeline

The parsing stage transforms a raw document file into a structured, typed content list. This list makes modality boundaries explicit and retains page-level positional information, which later stages use for context extraction and citation tracing.

3.1.1 Input Format Coverage

The pipeline accepts PDF, Microsoft Office formats (DOCX, PPTX, XLSX), HTML, plain text, Markdown, and images. Non-PDF inputs are normalized to PDF before parsing. Office and HTML documents are converted by an external LibreOffice installation. Plain text and

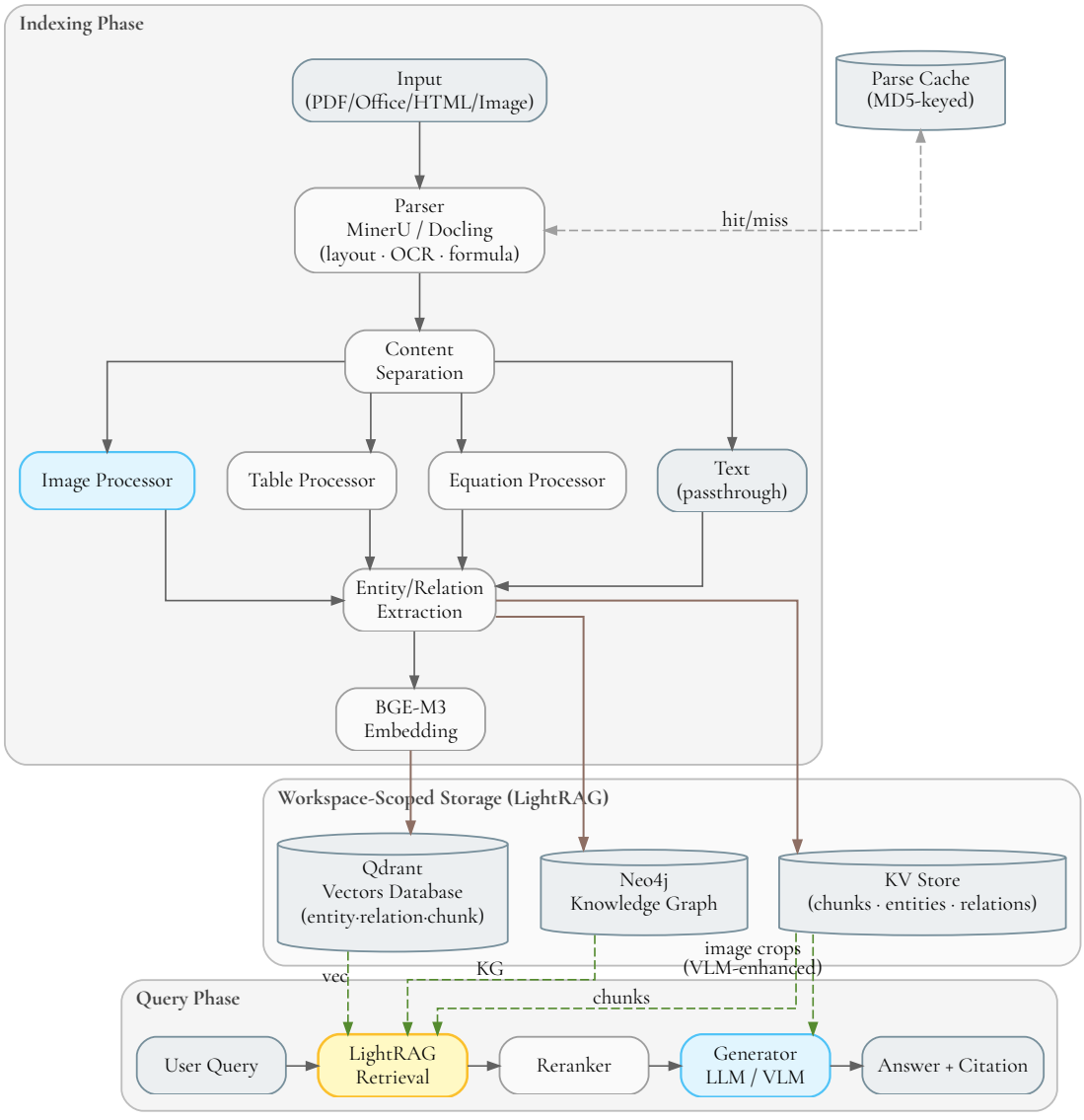


Figure 3.1: End-to-end pipeline of the multimodal RAG system.

Table 3.1: Schema of a parsed content record.

Field	Present in	Description
`type`	All	`text`, `image`, `table`, `equation`
`page_idx`	All	Zero-based page number
`text`	Text, equation	Extracted text or equation context
`img_path`	Image, table	Path to extracted image crop
`table_body`	Table	Structured cell text
`latex`	Equation	LaTeX string from formula recogniser

Markdown inputs are rendered to PDF through a fixed typographic template. This single-format normalization ensures that the downstream parser operates on a well-defined input regardless of the original source.

3.1.2 Layout-Aware Parsing with MinerU

We use MinerU 2.0 as the primary parsing backend (Wang et al., 2024a), which executes a multi-stage pipeline over individual PDF pages: a deep-learning layout model assigns bounding boxes and type labels (paragraph, title, figure, table, formula); detected regions are sorted into reading order based on bounding-box geometry; text regions are processed by an OCR engine; formula regions are converted to LaTeX strings by a dedicated recognition model; and table regions are processed by a structure recognition model that recovers the cell grid and serialises it as structured plain text. MinerU's *auto* mode applies OCR selectively based on whether the PDF is digitally born or scanned, minimising latency on native PDFs while remaining robust to scanned inputs.

The output is a list of typed content records whose schema is given in Table 3.1. Every record carries its modality type and page index, so that citations produced at query time can be resolved back to a specific page and modality in the original document.

3.1.3 Parse-Result Caching

Parsing is the most expensive ingestion step and is fully deterministic given the input bytes and parser configuration. The pipeline therefore persists parse results in a key–value namespace (`parse_cache`). The cache key is the MD5 digest of a configuration descriptor—parser backend, parse method, language, and device options—concatenated with a content fingerprint assembled, in reading order, from the text, image-path, table-body, and equation-text fields of each parsed record. On a cache hit, the stored content list is returned without re-invoking MinerU or any vision-language model, making incremental re-ingestion of updated document collections inexpensive even when only a small subset of documents has changed.

3.2 Multimodal Processing

The multimodal processing stage converts every non-text element into a natural-language description that can be embedded and retrieved by the text-based downstream components,

and simultaneously extracts a structured entity record that is inserted into the knowledge graph as a typed node. Converting non-text elements to text, rather than separate embeddings for each modality, allows the retrieval engine to operate on a uniform representation without separate indices per modality, and allows the knowledge graph to connect visual entities to text entities through shared relationship edges.

All modal processors inherit from a common base class that provides shared infrastructure for context extraction, JSON output parsing, chunk creation, and entity insertion. Three processors handle the canonical modalities, and a fourth handles the residual case.

Image processor. The image processor encodes the image crop as Base64 and constructs a multimodal VLM prompt requesting a self-contained description that covers visible components, spatial relationships, embedded text labels, and the connection to surrounding context. The response contains both a description and a structured entity record with name, type `image`, and summary.

Table processor. The table processor sends the serialised cell text together with the caption and, optionally, a rendering of the table image to the VLM. The prompt requests semantic interpretation: the table's purpose, the meaning of each column, notable data points, and the cross-row relationships a reader would need in order to interpret the data correctly.

Equation processor. The equation processor sends the LaTeX string to a LLM, requesting a natural-language rendering of the formula, identification of all variables and subscripts, and a statement of what the formula computes.

Generic processor. A fourth processor handles content items whose type falls outside the canonical {text, image, table, equation} set. It routes such items to the LLM with a schema-agnostic prompt and serves as the default fallback, ensuring that the pipeline degrades gracefully on unfamiliar inputs rather than dropping them silently.

Contextual grounding. Isolated non-text elements are often ambiguous without their surrounding context. Each processor is therefore supplied with a context window of adjacent text records drawn from the same and neighbouring pages, with section headers prioritised when the budget is tight. This allows the VLM to align entity names with the document's own terminology rather than generating generic labels, which in turn reduces synonym fragmentation in the downstream knowledge graph (Section 4.2).

Robust output parsing. Generative models occasionally produce malformed JSON. The base class implements a four-level fallback, direct extraction, tag stripping, quote repair, and regex-based field extraction, so that partial results are always recovered even when the model output is structurally defective.

VLM-as-captioner paradigm vs. vision-first embedding. The multimodal processing described above follows a VLM-as-captioner paradigm: non-textual elements are converted into textual descriptions by a vision language model, and these descriptions enter the same text-based retrieval and knowledge-graph construction pipeline as

ordinary text chunks. An alternative paradigm, exemplified by ColPali (Faysse et al., 2024), embeds entire document pages as images and retrieves directly in a visual embedding space, preserving fine-grained visual content without an intermediate text representation.

We adopt the VLM-as-captioner paradigm for two reasons. First, graph-based retrieval requires entities and relations as input; these can only be extracted from text, so captioning is a prerequisite for multimodal elements to participate in the knowledge graph and in PPR traversal (Chapters 4 and 5). Second, for the structured elements most common in academic and technical documents (tables with row–column semantics, mathematical formulas, and annotated diagrams) a textual representation generated by a capable VLM often preserves more retrievable semantic content than a fixed-dimensional visual embedding. The trade-off is that captioning is lossy at the *retrieval* stage: visual details not captured in the description are invisible to the embedding and graph indices, so queries that hinge on fine-grained visual features may fail to surface the relevant chunk. At the generation stage this loss is partially recovered, since the original image is re-injected into the VLM prompt as a Base64-encoded attachment alongside its caption, grounding the answer against the raw visual content.

3.3 Local Inference Stack

All models are served locally via vLLM, which exposes an OpenAI-compatible REST endpoint. This allows the pipeline to run entirely on-premise without sending any document content to external APIs, satisfying the privacy-preserving deployment requirement for proprietary industrial documents.

A single vision–language model (`Qwen3-VL-30B-A3B-FP8` in our best configuration) serves both indexing and query. Dense retrieval and reranking are handled by BGE-M3 (BAAI, 2024a; Chen et al., 2024) and bge-reranker-v2-m3 respectively, loaded locally through SentenceTransformers and FlagEmbedding. BGE-M3 produces 1,024-dimensional unit-norm dense vectors using FP16 inference where supported; the cross-encoder reranker scores query–document pairs and returns the top-n candidates sorted by descending relevance. Both encoders are instantiated once per server process at start-up and shared across all workspaces, which minimizes the GPU-memory cost.

3.4 Storage Architecture

Document data is partitioned into three independent directory trees, each persisting a different stage of the pipeline.

- **Uploads tier.** Stores original files exactly as uploaded. Served directly to the document viewer and never modified by the pipeline.
- **Output tier.** Stores MinerU parsing outputs: Markdown renderings, extracted image crops, and table screenshots. Used by the document reader and by the image-serving endpoint.
- **RAG workspace tier.** Stores the LightRAG working state, partitioned into three classes of persistent structure described in detail below: key–value stores, vector indices, and the knowledge graph.

The RAG workspace tier deserves further detail because it holds the structures consumed by every retrieval call.

Key–value stores. A set of JSON-backed key–value (KV) namespaces serves as the single source of truth for raw document content and extraction metadata. The pipeline maintains five namespaces: `full_docs` stores the full text of each ingested document, keyed by document identifier; `text_chunks` stores each chunk's text together with its token count, sequential position, source document identifier, and source file path, keyed by chunk identifier; `entities` and `relations` store the description, source chunk identifiers, and entity-type metadata produced by the extraction stage of Section 4.1.2, keyed by graph node and edge identifier respectively; `doc_status` tracks the per-document ingestion lifecycle (queued, processing, completed, failed). A sixth namespace, `parse_cache`, is shared across stages and described in Section 3.1; its keys are MD5 fingerprints of parser inputs, and its values are the cached MinerU content lists, allowing re-ingestion of unchanged documents to bypass parsing entirely.

The KV namespaces are not embedded and are not directly retrieved. They are accessed only by identifier, typically after a vector or graph hit has surfaced the corresponding identifier and the calling component needs the underlying text or metadata.

Vector indices. Dense-sparse embeddings are persisted in Qdrant, with three separate collections per workspace, one for chunks, one for entities, and one for relations, so that retrieval can target each level of granularity independently. Qdrant provides the ANN search underlying the vector retrieval channel.

Knowledge graph. The workspace knowledge graph is persisted in Neo4j Community Edition, which provides the indexed graph traversal and bulk adjacency export required by PPR retrieval. Both Neo4j and Qdrant are deployed as self-hosted services from publicly available release binaries, with no modifications to the upstream source.

The pipeline organises documents into *workspaces*: each upload is assigned an immutable `doc_id`, and a single `RAGAnything` instance is constructed per `workspace_id`, backed by a workspace-scoped working directory together with Neo4j and Qdrant namespaces. This gives strict isolation between workspaces, preventing cross-workspace knowledge graph contamination.

3.5 Web Interface

To demonstrate end-to-end usability, we implement a single-page web application that exposes the retrieval pipeline through a browser-based interface. The frontend is a React single-page application, and the backend is a FastAPI service that embeds the RAG pipeline as a Python module instantiated during the lifespan. The web service exposes endpoints covering four functions: document ingestion, querying, workspace management, and knowledge graph access.

Figure 3.2 shows the chat view in operation on a representative DocBench query. The user's question targets a specific numeric value (*67.5*) that appears only in a table cell of the

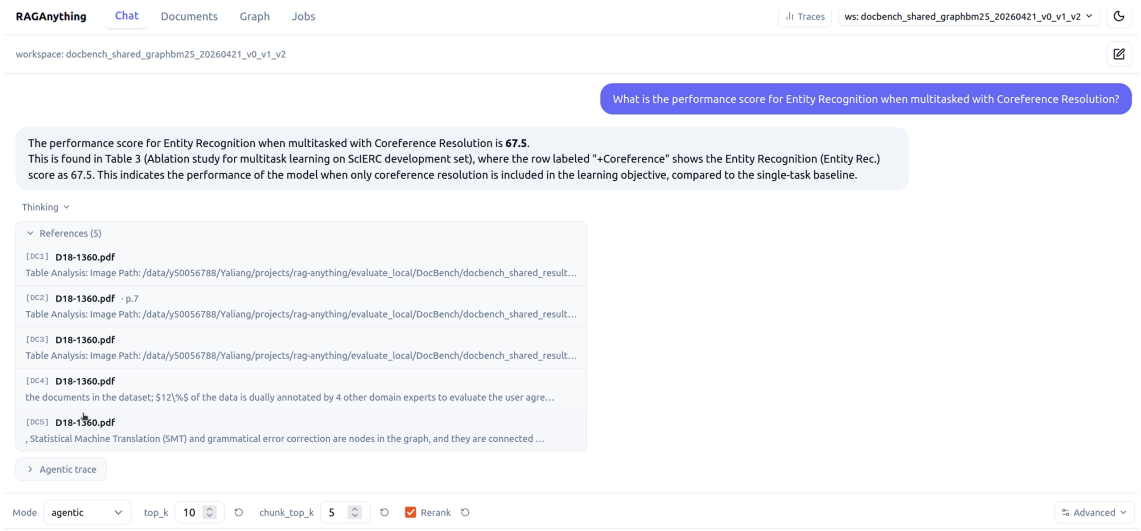


Figure 3.2: Chat view of the deployed web interface. The user asks a factual question requiring evidence located in a table; the generated answer cites the specific cell (*67.5*) and identifies its location (Table 3,"+Coreference" row). The *References* panel below the answer lists the chunks that were ultimately supplied to the generator as context; each entry is clickable and opens the source document at the originating passage. The *Agentic trace* panel (collapsed here, expanded in Figure 5.4) exposes the controller's routing, sufficiency, and grounding decisions.

source PDF; resolving it requires the multimodal indexing pipeline of Section 3.2 to have captured the table's row–column structure during ingestion. The generated answer is rendered together with the chunks that the agentic controller passed to the generator as final context; each reference entry is clickable and opens the source document at the originating passage, so any claim in the answer can be verified against the underlying chunk in a single interaction. Retrieval parameters (mode, top-k, chunk top-k, reranker toggle) are exposed as first-class UI controls.

Beyond querying, the interface exposes the constructed knowledge graph directly. Figure 3.3 shows the graph view for a workspace, in which nodes are colored by entity type and edges denote extracted relations; node search and inspection let a user trace how the answer-supporting entities are connected, making the graph construction (Chapter 4) inspectable.

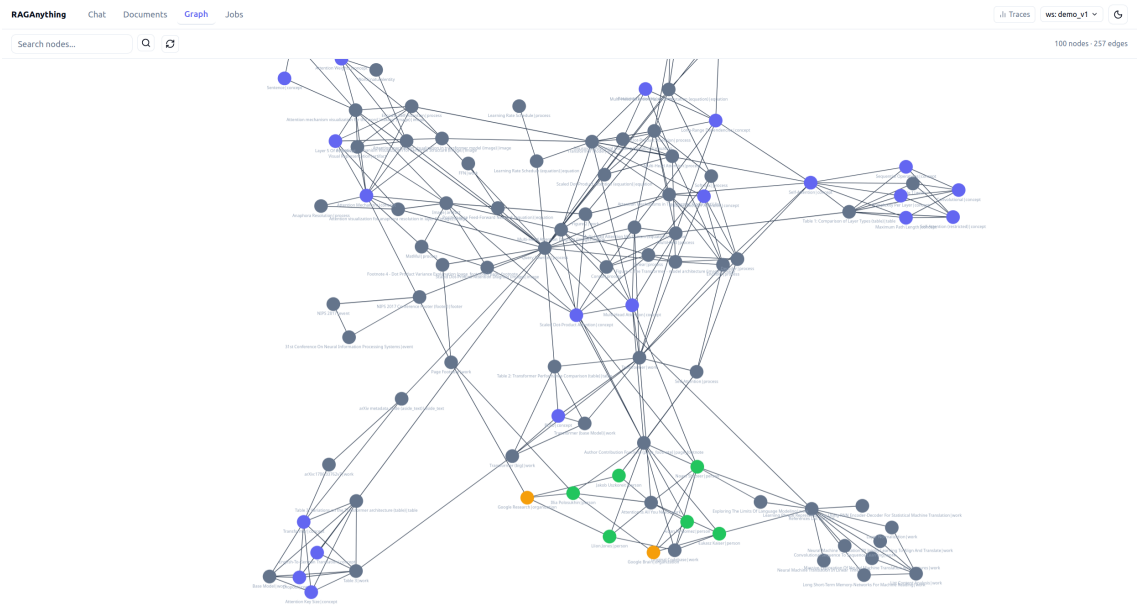


Figure 3.3: Knowledge graph view of the deployed web interface. Nodes represent extracted entities (colored by type) and edges represent relations. The search bar supports lookup of individual entities, and clicking a node expands its metadata and description.

Chapter 4
Knowledge Graph Construction

Chapter 3 described how documents are parsed and converted into unified textual representations. This chapter covers Stage 3 of the pipeline shown in Figure 3.1: how those representations are indexed into a knowledge graph, and how we optimize the resulting graph quality. Section 4.1 describes the LightRAG graph indexing process. Section 4.2 presents our KG optimization contributions.

4.1 Base Graph Indexing Pipeline

Once all document content has been converted to text—either through direct extraction for text regions or VLM-based description generation for non-text elements—the indexing stage builds the retrieval structures that support query-time evidence assembly. This stage is managed by LightRAG (Guo et al., 2025b), which maintains retrieval state at the workspace level rather than as one graph per document. A workspace may contain one or more documents.

The persistent backends of the indexing stage are described in Section 3.4. Briefly: chunk text and entity/relation metadata are written into the JSON-backed KV namespaces; entity, relation, and chunk embeddings are written into their respective Qdrant collections; and the knowledge graph itself is persisted in Neo4j. LightRAG's default backends (NanoVectorDB and an in-memory NetworkX graph) are overridden in our deployment to provide indexed graph traversal and approximate-nearest-neighbour search at workspace scale.

4.1.1 Text Chunking

Plain text content is split into overlapping token windows using LightRAG's built-in chunking module. Two parameters control the window: `chunk_token_size` (target token count per chunk) and `chunk_overlap_token_size` (tokens shared between consecutive chunks). Overlapping windows ensure that entities or relationships spanning a chunk boundary are captured in at least one complete chunk.

Each chunk is assigned a content-addressed identifier with the prefix `"chunk-"` followed by the MD5 hash of the chunk text. The chunk record is written into the `text_chunks` KV namespace (Section 3.4) with five fields: token count, full text, sequential position within the document, document ID, and source file path. The file path enables the citation system to surface the document, page, and modality of origin for every retrieved piece of evidence.

In our RAG-Anything-based implementation, VLM-generated descriptions of non-text elements are injected into the text pipeline as complete text chunks. We do not further split these descriptions, because each description is a self-contained summary of one visual or layout element. After this injection step, the descriptions enter the same chunk-level indexing pipeline as extracted document text.

4.1.2 Entity and Relationship Extraction

For each chunk, LightRAG performs entity and relationship extraction through a structured two-step process (Guo et al., 2025b).

Initial extraction

The LLM is prompted with a system instruction defining a knowledge graph specialist role and a user prompt containing the chunk text. The model is required to output all identified entities and relationships in a structured delimited format. Each entity record contains four fields: the record label `entity`, entity name, entity type, and a natural-language description. The entity name is generated according to the naming rules in the extraction prompt. The entity type is drawn from the predefined lowercase taxonomy used by our implementation: `person`, `organization`, `location`, `event`, `artifact`, `work`, `naturalentity`, `concept`, and `process`. Each relationship record contains five fields: the record label `relation`, source entity, target entity, relationship keywords, and a relationship description. Although the extraction format contains source and target fields, factual relations are canonicalized as undirected edges during graph insertion, so repeated mentions accumulate provenance rather than producing duplicate opposite-direction edges.

Supplementary extraction (gleaning)

After the initial extraction, LightRAG can perform a supplementary extraction pass, also referred to as gleaning. This pass uses the same chunk context and conversation history, but asks the model to identify entities and relationships that were missed or malformed in the initial output. The purpose is not to re-extract the whole chunk, but to recover incomplete records and improve coverage before graph construction.

When an entity or relationship is produced in both passes, the pipeline keeps the richer record when possible, for example the version with the more informative description. In this way, gleaning improves extraction recall while preserving the structured output format used by the downstream graph builder.

Output Parsing and Basic Cleanup

The extraction output uses a custom delimited text format rather than JSON. This design avoids common JSON-specific failures, such as nested quote escaping, incomplete brackets, and partially truncated objects. After generation, the pipeline splits the output into entity and relationship records, repairs corrupted delimiters, removes extraneous quotation marks, and validates entity-name length against a maximum character limit. This basic cleanup step improves the robustness of the extraction pipeline before graph construction.

For multimodal elements, entity records are produced during the description generation step in Section 3.2 and inserted directly into the graph as typed nodes. Specialized processors create nodes with entity types such as `image`, `table`, and `equation`. Other parser-level content types are handled by the generic processor and retain their original type labels, such as `page_number`, `footer`, `header`, `page_footnote`, or `ref_text`. These multimodal and layout-derived nodes participate in the same vector index as text-derived entities, allowing them to be retrieved alongside text evidence.

4.1.3 Entity Deduplication and Graph Merging

Multiple chunks may mention the same entity. In the baseline LightRAG indexing pipeline, extracted entity records are merged according to their graph node identifier, which is derived from the entity surface name. Records mapped to the same identifier are treated as the same graph node. Their descriptions, source chunk references, and file paths are aggregated into a single node record.

Relations are then written between the corresponding entity identifiers. Repeated mentions of the same relation accumulate provenance from multiple chunks rather than creating duplicate edges.

After graph merging, entities and relations are stored in the graph store for structural traversal. They are also embedded and stored in vector-indexed collections for semantic lookup.

4.1.4 Embedding and Vector Indexing

After text chunks, entity nodes, and relation records have been constructed, they are embedded and inserted into separate vector collections. LightRAG is model-agnostic with respect to the embedding function. The embedding model is passed at initialization, where the model name, output dimensionality, and maximum token budget are specified. In our deployment, we use BGE-M3 (BAAI, 2024a; Chen et al., 2024), which produces 1,024-dimensional unit-norm dense vectors and supports inputs up to 8,192 tokens. BGE-M3 was selected for its multilingual capability, strong performance on long technical documents, and support for dense, sparse, and multi-vector retrieval modes.

LightRAG's default vector backend is NanoVectorDB, a lightweight local vector store that persists embeddings as JSON files. In our local deployment, we instead use Qdrant as the vector backend while preserving LightRAG's logical separation between chunk, entity, and relation indices. Separate vector collections are maintained for chunks, entities, and relationships, allowing targeted retrieval at each level of granularity.

4.2 Graph Quality Optimization

The base LightRAG indexing pipeline constructs graph nodes and edges from LLM-extracted entity and relationship records. In practice, we observed three graph-quality issues.

First, entity surface forms are not always consistent across chunks or between entity and relationship records. Small variations in casing, punctuation, separators, whitespace, or intra-word hyphens can cause the same entity to be inserted as multiple graph nodes, which fragments entity-centered evidence.

Second, relation endpoints are generated separately from entity records. An endpoint mentioned in a relationship record may not have been extracted as a normal entity record in the same scope. If such a relation is inserted without validation, the missing endpoint can lead to a fallback node with an `UNKNOWN` type. This node is not a normal extracted entity; its content is derived from relation text rather than from a dedicated entity description, creating a relation-only pseudo-entity.

Third, name-based merging can collapse polysemous entities that share the same surface form, while equivalent entities expressed through different surface forms can remain disconnected.

We address these issues with three lightweight modifications. Surface normalization reduces fragmentation caused by minor surface-form variation. Strict endpoint validation removes relation records whose endpoints cannot be safely matched to extracted entity nodes. Type-aware entity identifiers reduce erroneous merging of polysemous entities. Synonym linking adds synthetic edges between semantically equivalent surface forms without merging their nodes.

4.2.1 Surface Normalization and Endpoint Validation

Before graph insertion, entity names and relation endpoints are passed through the same surface-normalization routine. The routine applies Unicode NFKC normalization, removes separator artifacts, collapses whitespace, normalizes intra-word hyphens, and applies rule-based casing. Domain acronyms in an uppercase allowlist, such as `AI`, `LLM`, `OCR`, `RAG`, `GPU`, and `JSON`, are preserved.

Before endpoint matching, relation records are discarded if either normalized endpoint is empty or if the two endpoints become identical after normalization. We then apply strict endpoint matching during graph insertion: a relation is written only if both normalized endpoints match existing entity nodes in the current endpoint scope. This prevents dangling relation edges and avoids creating relation-only pseudo-entities from relation endpoints that are not supported by extracted entity records.

4.2.2 Entity Disambiguation

In the baseline implementation, the graph node key and vector database identifier for each entity are derived from the entity name alone:

$$\texttt{graph\_key} = \texttt{entity\_name} \tag{4.1}$$

$$\texttt{vdb\_id} = \texttt{"ent-"} \,\|\, \mathrm{MD5}(\texttt{entity\_name}) \tag{4.2}$$

Entity type is extracted during the knowledge graph extraction step but is discarded before identifier computation. As a result, polysemous entities are assigned identical identifiers regardless of their semantic context, causing them to be merged into a single graph node. Merged nodes accumulate contradictory descriptions from distinct entity senses, degrading both embedding quality and retrieval precision.

To address this problem, we extend identifier computation to incorporate entity type as a disambiguating dimension. The graph node key becomes the concatenation of entity name and normalized entity type:

$$\texttt{graph\_key} = \texttt{entity\_name} \parallel \texttt{"|"} \parallel \texttt{entity\_type} \tag{4.3}$$

The vector database identifier is updated accordingly:

$$\texttt{vdb\_id} = \texttt{"ent-"} \parallel \mathrm{MD5}(\texttt{entity\_name} \parallel \texttt{"|"} \parallel \texttt{entity\_type}) \tag{4.4}$$

Entity type is normalized to lowercase with whitespace removed before concatenation, consistent with the normalization applied during extraction. For example, *Amazon* as a company is keyed as `"Amazon|organization"`, while *Amazon* as a river or geographic region is keyed as `"Amazon|location"`. Each sense is represented as an independent graph node with its own description, relationship edges, and source chunk references.

This modification requires no changes to the extraction prompt or downstream retrieval logic. Entity type is already produced during extraction; we include it in the identifier rather than discarding it. The trade-off is a modest increase in graph size from separating previously merged polysemous nodes. In domain-specific technical documents, polysemous entities are relatively rare, so the increase in practice is small.

4.2.3 Synonym Linking

Surface-form variation is common in technical and multilingual documents. Abbreviations, transliterations, and cross-lingual equivalents—such as *AI* and *Artificial Intelligence*, or *Beijing* and *Peking*—may denote the same concept but appear as disconnected nodes in the knowledge graph. Queries targeting one surface form may therefore fail to reach evidence associated with another surface form, reducing recall on semantically equivalent entities.

To address this problem, we add synthetic synonym edges after factual graph construction is complete for a workspace. The procedure follows the synonym-graph idea of HippoRAG 2 (Gutiérrez et al., 2025), but adapts it to our LightRAG/RAG-Anything storage layout. Entity nodes are not merged. Instead, semantically similar entity nodes remain separate and are connected by undirected SYNONYM edges. This preserves the original provenance and avoids collapsing potentially distinct entity senses.

The synonym-linking step first retrieves the completed set of entity labels from the workspace knowledge graph and fetches their corresponding entity embeddings from the entity vector store. Candidate source entities are filtered by effective character count, where punctuation and whitespace are ignored, to prevent very short strings from becoming noisy synonym hubs. Pairwise cosine similarity is then computed over the entity embeddings, using the fact that BGE-M3 embeddings are L2-normalized. Entity pairs whose similarity exceeds the threshold $\tau_{\text{syn}} = 0.8$ are considered synonym candidates.

For each candidate pair, the endpoints are canonicalized before insertion so that the undirected synonym relation is written only once. Existing factual relation edges are preserved: a

synthetic SYNONYM edge is inserted only when it does not overwrite a factual edge between the same endpoints. Existing synthetic synonym edges can be cleared and rebuilt when the threshold or graph content changes, but factual edges are never removed by this operation.

The resulting graph contains both factual edges extracted from document chunks and synthetic synonym edges derived from entity embeddings. During PPR-based retrieval, these synonym edges allow relevance to propagate across surface-form variants. For example, a query mentioning *AI* can reach evidence indexed under *Artificial Intelligence* through a single SYNONYM hop, potentially recovering evidence that name-based graph traversal or sparse matching would otherwise miss. Since synonym linking adds edges rather than merging nodes, it improves graph connectivity while keeping entity identity and source provenance explicit.

Chapter 5
Enhanced Retrieval

Chapter 4 described how we construct and optimize the knowledge graph. This chapter addresses the retrieval layer. Section 5.1 identifies the limitations of LightRAG's default retrieval mechanism. Section 5.2 presents our PPR-based multi-hop retrieval extension.

5.1 Limitations of LightRAG Retrieval

LightRAG's default retrieval operates as a dual-level lookup over the knowledge graph and vector indices, as described in Section 4.1. At query time, high-level and low-level keywords are extracted from the query and used to retrieve matching entities and relations via vector similarity. Retrieved entities and their immediate neighbors, retrieved relationships and their source and target nodes, and the associated chunks of them are assembled as context for the generator. This design has two structural limitations in the multimodal knowledge graph setting.

One-hop constraint Retrieval is limited to directly matched entities, relationships and their immediate neighbors. Queries that require reasoning across multiple relational steps—for example, connecting a figure to the methodology it illustrates, and from there to the quantitative result it supports—cannot be answered from a one-hop neighborhood. The relational paths encoded in the knowledge graph are therefore underutilized.

Entity-to-chunk indirect mapping In the default pipeline, retrieved entities are used to look up their associated source chunks via a secondary index. This two-stage mapping introduces a disconnect between graph-based relevance signals and the chunks ultimately presented to the generator, since chunk relevance is not directly optimized during graph traversal.

The remaining sections address these limitations.

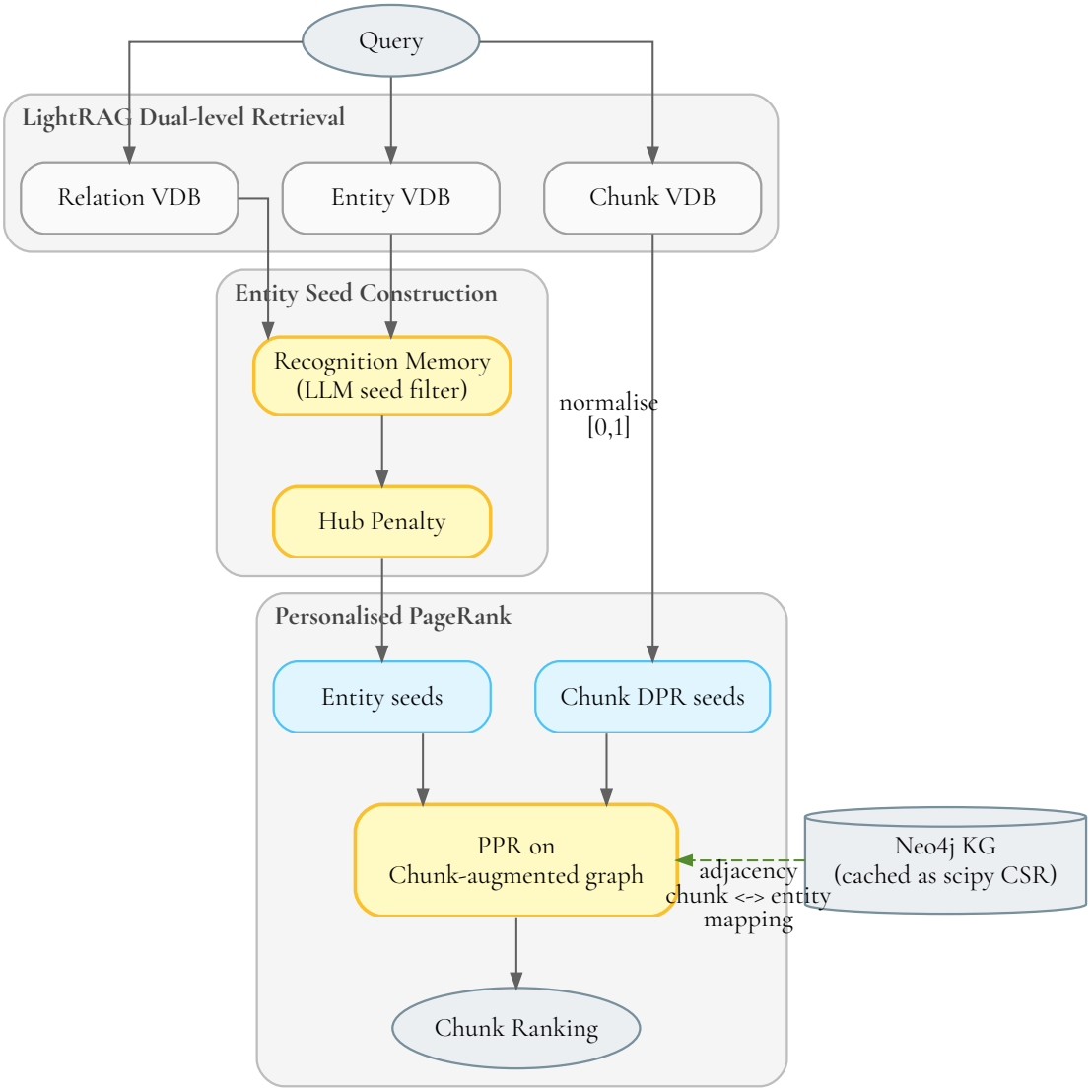


Figure 5.1: The PPR-based multi-hop retrieval pipeline.

5.2 PPR-based Multi-hop Retrieval

5.2.1 Overview

We extend LightRAG's one-hop retrieval with PPR (Gutiérrez et al., 2025) over the knowledge graph. PPR assigns a relevance score to every node in the graph by simulating a biased random walk that restarts at a set of query-specific seed nodes with probability α at each step. Nodes structurally proximate to highly weighted seeds receive high scores even without a direct semantic match to the query embedding. Critically, document chunks are modeled as first-class nodes in an augmented graph rather than as secondary artefacts of entity retrieval, so PPR produces chunk-level relevance scores directly without requiring a post-hoc entity-to-chunk projection.

5.2.2 Seed Initialization

The personalization vector $\mathbf{s} \in \mathbb{R}^{N+M}$ over the N entity nodes and M chunk nodes is constructed from two independent signals: entity seeds, which inject query-relevant graph anchors, and chunk seeds, which inject direct chunk-level evidence. We describe each in turn, then specify how they are combined.

Throughout this section, all min-max and ℓ_1 normalizations are applied to the support of the corresponding signal—that is, only over the non-zero entries—rather than over the full $N + M$-dimensional vector. Nodes outside the support receive a seed weight of zero but remain active PPR nodes and can accumulate relevance through graph propagation.

Entity Seed Construction

Entity seed construction proceeds in four stages: (i) dual-level candidate retrieval, (ii) recognition memory filtering, (iii) score computation by averaging and max-merge on the recognized set, and (iv) hub penalization. The result is an unnormalized entity weight vector supported on a small subset of V_e.

Stage 1: Dual-level candidate retrieval. Following LightRAG (Guo et al., 2025b), the input query is first passed to an LLM that extracts two complementary keyword sets aligned with two distinct retrieval channels:

- *Low-level keywords* name specific entities or concepts mentioned in the query. They are embedded into a vector $\mathbf{q}_{\text{low}}$ and used to query the entity vector index $\mathcal{V}_e$, whose records are entity-description embeddings. This channel surfaces candidate entities through direct name-and-description matching.
- *High-level keywords* capture the abstract themes, relations, or topics of the query. They are embedded into a vector $\mathbf{q}_{\text{high}}$ and used to query the relation vector index $\mathcal{V}_r$, whose records are relation-fact embeddings. This channel surfaces candidate entities through the relational facts they participate in.

Each channel returns the top-k_{cand} records with cosine similarity scores. The two indices play distinct roles. The entity index directly yields a set of candidate entities C_e with raw similarity scores $s_{\text{entity}}(e)$. The relation index yields a set of relation records C_r with similarity scores $s(r)$; for each retrieved relation, both its subject and object entity identifiers are extracted as additional seed candidates. The relation index therefore acts as a second retrieval channel that surfaces entities through the relational facts they participate in, complementing the direct entity-description channel.

The output of this stage is a candidate pool $C = C_e \cup \{\text{endpoints}(r) : r \in C_r\}$, together with each candidate's raw similarity evidence from both channels.

Stage 2: Recognition memory filtering. The candidate pool produced by Stage 1 contains entities that are semantically related to the query but are not necessarily informative anchors for answering it—for example, generic concepts or topically broad entities that surface in vector retrieval through embedding-level affinity alone. Following HippoRAG 2 (Gutiérrez et al., 2025), we apply a recognition memory step to filter the pool by content relevance before any score processing takes place.

The candidate pool is serialized as a list of entity descriptions together with relational triples in the form ⟨src | relation | tgt⟩, and presented to an LLM with a recall-oriented prompt that asks it to identify the entity identifiers directly relevant to answering the query. The LLM returns a subset of identifiers, which are matched back to C via fuzzy string matching to tolerate minor formatting drift while rejecting confabulated identifiers absent from the pool. The output is the recognized entity set $\mathcal{E}_{\text{rec}} \subseteq C$, typically of size five.

The LLM judges relevance from entity descriptions and relational triples—textual content—rather than from numerical scores. If the LLM call fails or returns an empty result, the system falls back to using the full candidate pool C without filtering, preserving retrieval continuity.

Stage 3: Score computation on the recognized set. For each recognized entity $e \in \mathcal{E}_{\text{rec}}$, we compute a single relevance weight by aggregating the evidence from the two retrieval channels.

First, when e appears as an endpoint across multiple retrieved relations $\mathcal{R}_e = \{r \in C_r : e \in \text{endpoints}(r)\}$, its relation-derived score is averaged:

$$s_{\text{relation}}(e) = \frac{1}{|\mathcal{R}_e|} \sum_{r \in \mathcal{R}_e} s(r), \tag{5.1}$$

and is set to 0 if $\mathcal{R}_e = \emptyset$. This averaging, analogous to the occurrence-count normalization in HippoRAG 2 (Gutiérrez et al., 2025), prevents entities that happen to appear in many low-scoring relations from accumulating spurious mass; entities supported by a small number of high-scoring relations are not penalized.

Second, the two channel signals are merged by element-wise maximum:

$$w_0(e) = \max(s_{\text{entity}}(e),\ s_{\text{relation}}(e)), \tag{5.2}$$

with $s_{\text{entity}}(e) = 0$ when $e \notin C_e$. Both channels use the same BGE-M3 embedding space and produce cosine similarity scores on a comparable scale, so the max can be taken on the raw scores directly without per-source min-max normalization. Max-merge reflects the semantics that the two sources are alternative evidence channels for the same proposition (*this entity is relevant*): an entity strongly supported by either channel should receive full weight without being diluted by the other channel's potentially zero contribution.

The output of this stage is an unnormalized weight $w_0(e) \geq 0$ defined on the recognized set.

Stage 4: Hub penalization. A highly connected entity that spans many unrelated topics can still survive recognition memory—the LLM judges semantic relevance, not corpus-wide specificity. Such entities can dominate the PPR initialization and draw random-walk probability away from more specific seeds. We discount them by dividing their weight by the number of distinct chunks in which the entity is mentioned, $|C(e)|$:

$$\tilde{w}_{\text{entity}}(e) = \begin{cases} w_0(e)/|C(e)| & \text{if } |C(e)| > \delta_{\text{hub}}, \\ w_0(e) & \text{otherwise}, \end{cases} \tag{5.3}$$

where δ_{hub} is the threshold below which no penalty is applied. The denominator $|C(e)|$ quantifies corpus-wide specificity directly: an entity appearing in an excessive number of chunks

is treated as a low-information connector, regardless of its topological richness in the local graph structure.

The penalized weights are then ℓ_1-normalized over the recognized set, producing the final entity seed sub-vector:

$$w_{\text{entity}}(e) = \frac{\tilde{w}_{\text{entity}}(e)}{\sum_{e' \in \mathcal{E}_{\text{rec}}} \tilde{w}_{\text{entity}}(e')}, \qquad e \in \mathcal{E}_{\text{rec}}. \tag{5.4}$$

This normalization gives the entity sub-vector total mass 1, allowing it to be composed with the chunk sub-vector at a fixed entity-to-chunk mass ratio of $1{:}p$, independent of the absolute magnitude of the raw cosine and hub-penalized scores.

Chunk Seed Construction

With the entity and chunk sub-vectors each internally normalized, the final step is to assemble them into a single personalization vector over the full augmented node set.

In parallel with the entity branch, a chunk-level signal supplies a direct chunk anchor for the random walk. A dense passage retrieval (DPR) query over the chunk vector index returns the top $4k_{\text{PPR}}$ chunks with similarity scores. The factor of four is a coverage argument: the chunk seed pool must be larger than the final PPR output window k_{PPR} so that directly query-relevant passages enter the random walk as initialization points rather than being discovered only through entity-mediated propagation.

The retrieval scores are min-max normalized over the candidate pool to $\hat{s}(c) \in [0, 1]$, then normalized to unit ℓ_1 norm so that the candidate pool itself forms a probability distribution, and finally scaled by a global mass factor p:

$$w_{\text{chunk}}(c) = p \cdot \frac{\hat{s}(c)}{\sum_{c'} \hat{s}(c')}, \qquad c \in \text{top-}4k_{\text{PPR}}. \tag{5.5}$$

We use $p = 0.05$, following HippoRAG 2. The min-max step standardizes the scale of the raw retrieval scores; the subsequent ℓ_1 normalization converts them into a relative-mass distribution; the factor p then sets the total mass of the chunk-seed sub-vector to 0.05. The total seed mass allocated to entities and chunks is therefore in a fixed $1{:}p$ ratio, independent of either branch's score scale.

Combined Seed Vector

The entity and chunk weight vectors are combined into a single personalization vector over the full augmented node set:

$$\mathbf{s}[e] = w_{\text{entity}}(e), \qquad \mathbf{s}[c] = w_{\text{chunk}}(c). \tag{5.6}$$

The entity sub-vector is ℓ_1-normalized over the recognized entity set so that its total mass is 1. The chunk sub-vector has already been scaled to total mass p. The composite vector therefore has ℓ_1 norm $1 + p$ and is finally re-normalized to unit ℓ_1 norm before power iteration:

$$\mathbf{s} \leftarrow \frac{[w_{\text{entity}};\, w_{\text{chunk}}]}{1 + p}. \tag{5.7}$$

After this final normalization, the entity sub-vector carries a total mass of $1/(1+p) \approx 0.952$ and the chunk sub-vector carries $p/(1+p) \approx 0.048$. The fixed allocation reflects a design choice: entity seeds carry the primary semantic injection signal, while chunk seeds provide a small residual mass that guarantees direct passage-level anchors enter the random walk independently of the entity graph topology.

5.2.3 Chunk-Augmented Graph Construction

Document chunks are incorporated into the graph as virtual nodes, extending the post-resolution entity–entity adjacency structure. For each chunk c, edges are added between c and every entity e whose extraction in the knowledge graph cites c as a source, i.e., whose source id field references the chunk identifier. This connectivity is derived from both node-level and edge-level provenance: an entity node e is connected to all chunks in its source id list, and a relation edge (e_i, e_j) additionally contributes connections between its source chunks and both of its endpoint entities. Formally, let $\text{prov}(e)$ denote the chunk-id set in e's source id field and $\text{prov}(r)$ the chunk-id set of relation r; the chunk–entity edge set is

$$E_{ce} = \{(e,c) : c \in \text{prov}(e)\} \cup \{(e,c) : \exists\, r = (e,e') \in E_e \text{ with } c \in \text{prov}(r)\}, \tag{5.8}$$

and the full augmented edge set is $E_{\text{aug}} = E_e \cup E_{ce}$.

The corresponding sparse adjacency matrix $\mathbf{A} \in \mathbb{R}^{(N+M)\times(N+M)}$ indexes both entity and chunk nodes in a unified node ordering $V_{\text{aug}} = V_e \cup V_c$:

$$\mathbf{A}_{ij} = \begin{cases} w(e_i, e_j) & \text{if } (e_i, e_j) \in E_e, \\ 1 & \text{if } (i,j) \in E_{ce} \text{ or } (j,i) \in E_{ce}, \\ 0 & \text{otherwise,} \end{cases} \tag{5.9}$$

where $w(\cdot,\cdot)$ denotes the stored relation weight and the unit weight on chunk–entity edges follows the convention discussed below. All chunk–entity edges carry a uniform weight of 1.0, matching HippoRAG 2's (Gutiérrez et al., 2025) passage–entity edge convention. Entity–entity edges carry their stored graph weights, which reflect the accumulated evidence strength of the corresponding relation. This construction spans the entire knowledge graph: all stored chunk identifiers are included as virtual PPR nodes, giving the random walk access to the complete document collection.Chunk–chunk edges are not constructed; chunks reach one another only through shared entity neighbours. This design both keeps the graph signal disjoint from the dense retrieval signal already injected via chunk seeds and ensures that any two chunks are connected only via an entity the extraction pipeline identified in both, so the propagation path itself encodes *why* the chunks are related. Chunk embedding similarity is not used to build edges in the augmented graph. Multimodal entities typed as `image`, `table`, and `equation` (Section 3.2) participate in V_e on equal footing with text-derived entities, so multimodal evidence is reachable from textual queries via the same traversal.

5.2.4 PPR Computation

PPR is computed by power iteration over the chunk-augmented adjacency matrix. Both entity–entity and chunk–entity edges are explicitly symmetrized at construction time: every

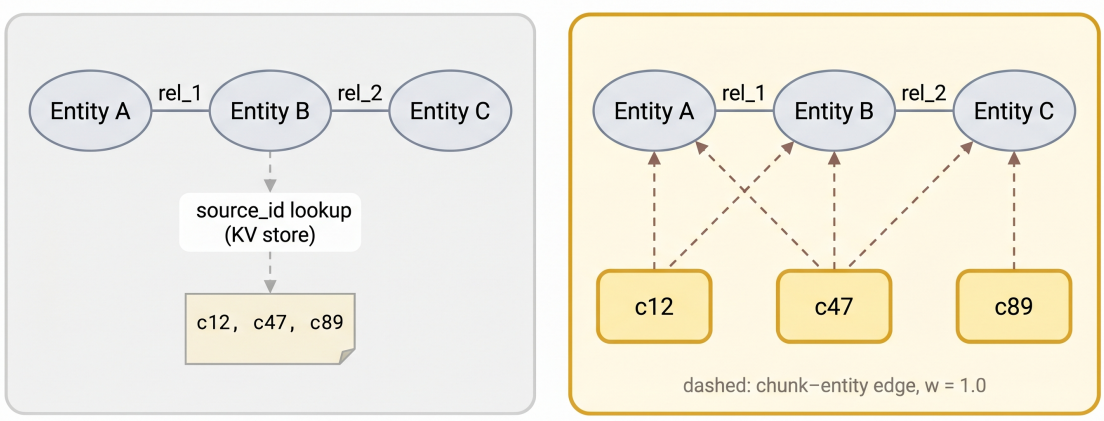


Figure 5.2: Graph schema comparison. In LightRAG (a), chunks are not graph nodes: each entity record stores the identifiers of the source chunks from which it was extracted, and chunk-level retrieval must be reconstructed at query time through an entity-to-chunk projection. In the chunk-augmented schema adopted by this thesis (b), chunks become first-class nodes connected to entities through MENTIONS edges, allowing Personalised PageRank to propagate relevance directly to chunk-level scores without an intermediate projection step.

edge (i, j) with weight w contributes two equal entries $\mathbf{A}_{ij} = \mathbf{A}_{ji} = w$ in the sparse CSR adjacency matrix, so the graph fed to PPR is undirected and weighted.

Let $\mathbf{A}$ denote the adjacency matrix of the augmented graph with N entity nodes and M chunk nodes. The transition matrix $\mathbf{P}$ is obtained from $\mathbf{A}$ by row-stochastic normalization: row i is divided by its row sum, so that $\mathbf{P}_{ij}$ is the probability of stepping from node i to node j. Dangling rows—nodes whose row sum is zero—are redirected to the personalization vector $\mathbf{s}$, which preserves total probability mass under iteration and prevents seed-disconnected nodes from acting as probability sinks. We use this row-stochastic form directly rather than the symmetric normalization $\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$ common in spectral graph methods, because $\mathbf{P}$ admits a direct random-walk interpretation that matches the PPR semantics, whereas the symmetric form does not.

Given the personalization vector $\mathbf{s}$ defined in Section 5.2.2, the PPR distribution $\boldsymbol{\pi} \in \mathbb{R}^{N+M}$ is the unique fixed point of

$$\boldsymbol{\pi} = (1 - \alpha)\,\mathbf{P}^{\top}\boldsymbol{\pi} + \alpha\,\mathbf{s}, \tag{5.10}$$

where $\alpha = 0.5$ is the restart (teleport) probability, following the original HippoRAG 2 design (Gutiérrez et al., 2025). The right-hand side is a contraction with factor $(1 - \alpha)$ in ℓ_1, so the fixed point exists, is unique, and is reached by power iteration starting from $\boldsymbol{\pi}^{(0)} = \mathbf{s}$:

$$\boldsymbol{\pi}^{(t+1)} = (1 - \alpha)\,\mathbf{P}^{\top}\boldsymbol{\pi}^{(t)} + \alpha\,\mathbf{s}, \tag{5.11}$$

terminated when $\|\boldsymbol{\pi}^{(t+1)} - \boldsymbol{\pi}^{(t)}\|_1 < 10^{-6}$. We use the `fast-pagerank` implementation, which executes each step as a sparse matrix–vector product against the cached CSR adjacency, with

Algorithm 1 PPR-based Multi-hop Chunk Retrieval

Require: Dual-level query embeddings $\mathbf{q}_{\text{low}}, \mathbf{q}_{\text{high}}$; VDBs $\mathcal{V}_e, \mathcal{V}_r, \mathcal{V}_c$; cached augmented adjacency $\mathbf{A}_{\text{aug}}$; chunk frequencies $|\mathcal{C}(\cdot)|$; hyperparameters $\alpha, \delta_{\text{hub}}, p, k_{\text{cand}}, k_{\text{recog}}, k_{\text{PPR}}$
Ensure: Ranked list of retrieved chunks $\mathcal{C}_{\text{PPR}}$

— Stages 1–2: Candidate retrieval and recognition memory —
$\mathcal{C}_e \leftarrow \textsc{TopK}(\mathcal{V}_e, \mathbf{q}_{\text{low}}, k_{\text{cand}})$; $\mathcal{C}_r \leftarrow \textsc{TopK}(\mathcal{V}_r, \mathbf{q}_{\text{high}}, k_{\text{cand}})$
$\mathcal{C} \leftarrow \mathcal{C}_e \cup \{\textsc{Endpoints}(r) : r \in \mathcal{C}_r\}$
$\mathcal{E}_{\text{rec}} \leftarrow \textsc{RecognitionMemory}(\mathcal{C}, k_{\text{recog}})$, **fallback** $\mathcal{C}$ on failure
— Stages 3–4: Score computation, hub penalty, normalization —
for each $e \in \mathcal{E}_{\text{rec}}$ **do**
    $\mathcal{R}_e \leftarrow \{r \in \mathcal{C}_r : e \in \textsc{Endpoints}(r)\}$
    $s_{\text{rel}}(e) \leftarrow \frac{1}{|\mathcal{R}_e|} \sum_{r \in \mathcal{R}_e} s(r)$ **if** $\mathcal{R}_e \neq \emptyset$ **else** 0
    $w(e) \leftarrow \max(s_{\text{entity}}(e),\ s_{\text{rel}}(e))$
    $w(e) \leftarrow w(e)/|\mathcal{C}(e)|$ **if** $|\mathcal{C}(e)| > \delta_{\text{hub}}$ ▷ Hub penalty
end for
$w_{\text{entity}}(e) \leftarrow w(e) \,/\, \sum_{e' \in \mathcal{E}_{\text{rec}}} w(e')$ ▷ ℓ_1-normalize entity sub-vector
— Chunk seed and combined personalization vector —
$\mathcal{C}_{\text{seed}} \leftarrow \textsc{TopK}(\mathcal{V}_c, \mathbf{q}_{\text{low}}, 4k_{\text{PPR}})$; apply min-max to scores $\hat{s}(c)$
$w_{\text{chunk}}(c) \leftarrow p \cdot \hat{s}(c) / \sum_{c'} \hat{s}(c')$
$\mathbf{s} \leftarrow [w_{\text{entity}};\ w_{\text{chunk}}]/(1+p)$ ▷ Combined and final ℓ_1-normalized
— PPR power iteration —
$\boldsymbol{\pi}^{(0)} \leftarrow \mathbf{s}$; $\mathbf{P} \leftarrow \textsc{RowNormalize}(\mathbf{A}_{\text{aug}})$ ▷ Dangling rows fall back to $\mathbf{s}$
repeat
    $\boldsymbol{\pi}^{(t+1)} \leftarrow (1-\alpha)\,\mathbf{P}^{\top}\boldsymbol{\pi}^{(t)} + \alpha\,\mathbf{s}$
until $\|\boldsymbol{\pi}^{(t+1)} - \boldsymbol{\pi}^{(t)}\|_1 < 10^{-6}$
return $\textsc{TopK}(\boldsymbol{\pi}[\text{chunk nodes}],\ k_{\text{PPR}})$

per-step cost $O(|E_{\text{aug}}|)$ in the number of augmented edges. On our DocBench graph (approximately 1.9×10^4 nodes, 7.5×10^4 edges), convergence requires 30–50 iterations and the entire PPR call completes in around 100 ms.

The chunk-level relevance scores are then read off from the entries of $\boldsymbol{\pi}$ corresponding to chunk nodes, sorted in descending order, and truncated to the top k_{PPR} chunks as the PPR retrieval output.

5.3 Agentic RAG

The retrieval components described in the preceding sections—PPR graph traversal, RAG-Anything knowledge-graph retrieval, and sparse–dense vector retrieval—are coordinated by a closed-loop agentic pipeline. The goal is to address common failures in single-pass retrieval: selecting an unsuitable retrieval strategy, generating an answer from insufficient evidence, and returning hallucinated claims without verification.

The pipeline is implemented as a deterministic finite-state controller with five main components: an adaptive router, a retriever, a sufficiency grader, a generator, and a hallucination verifier. Two auxiliary components, a query rewriter and a query decomposer, are invoked

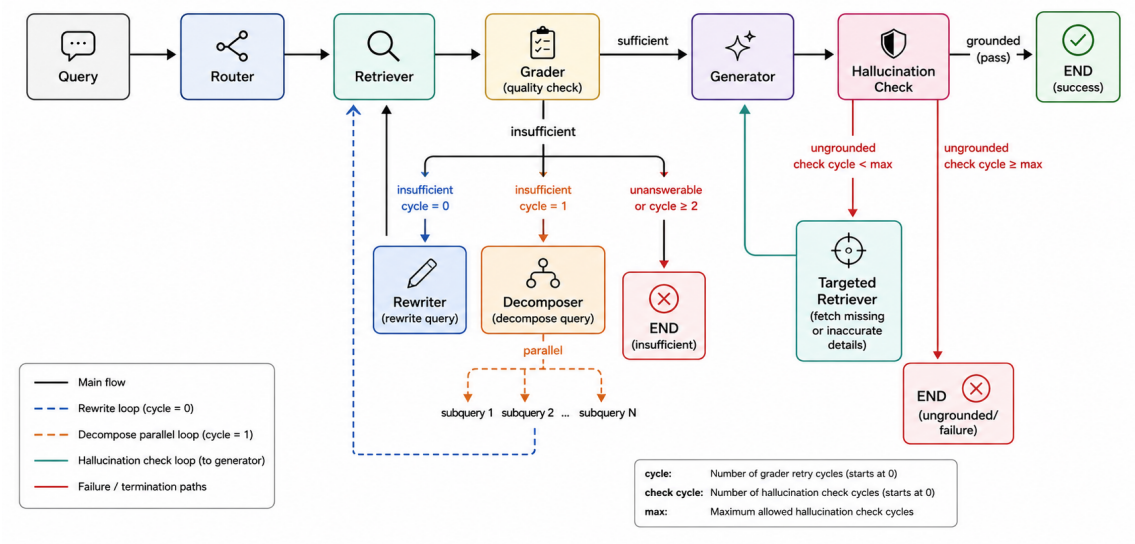


Figure 5.3: The agentic retrieval pipeline expressed as a deterministic finite-state controller.

only during failure-driven escalation. We use LangGraph (LangChain Team, 2024) to express the controller as a typed state graph: each component is a node, transitions are conditional edges driven by sufficiency and grounding signals, and the shared state object carries the active query, the accumulated retrieval set, and the recovery-budget counters. The prompted controller components and the generator share the same locally deployed VLM. Retrieval itself relies on the local graph store, vector store, embedding model, and, when enabled by the selected profile, the reranker.

The experiments in this thesis use the `agentic` controller, which follows a fixed three-round retrieval procedure. A separate `agentic_v2` recovery-policy variant exists in the codebase, but it is not used for the results reported in Chapter 6.

5.3.1 Adaptive Query Routing

The router maps an incoming query to one of five routing profiles:

$$\pi \in \Pi_{\text{route}} = \{\textsc{precise}, \textsc{semantic}, \textsc{local}, \textsc{global}, \textsc{multihop}\}.$$

Each profile selects a retrieval strategy suited to a different query type. Weighted RRF is applied when a selected profile or a recovery profile activates one or more ranked retrieval paths.

Precise activates sparse lexical retrieval only. It is used for queries with hard lexical constraints, such as error codes, identifiers, version numbers, exact titles, quoted strings, table names, code names, or rare strings that must match the document text. A person, organization, or location name alone is not sufficient to trigger this profile.

Semantic activates sparse–dense hybrid chunk retrieval without graph traversal. It covers ordinary factual, definitional, summary, and procedural questions where graph structure is not clearly needed.

Local activates local knowledge-graph retrieval around a focal entity and its direct attributes, neighbours, or relationships. It is used when the question concerns one central entity and its immediate relational context.

Global activates relation-centric knowledge-graph retrieval. It is used for questions about relationships, events, themes, participation, interaction, affiliation, dependency, or other cases where the queried relation is more important than a single focal entity.

Multihop activates PPR traversal over the chunk-augmented knowledge graph. Sparse–dense vector retrieval is used inside the PPR path for seed construction, but it is not treated as a separate fused retrieval path in this routing profile. This profile targets bridge, comparison, causal-chain, shared-intermediate, and cross-document questions where random-walk propagation can surface indirectly connected evidence.

Full recovery profile Full is not emitted by the router. It is invoked during failure escalation after routed retrieval remains insufficient. In our implementation, it combines PPR retrieval, sparse–dense hybrid chunk retrieval, and sparse lexical retrieval through weighted RRF, with a higher weight assigned to the PPR path. This gives the controller a terminal high-recall recovery path.

Routing is performed by the VLM using a prompt that lists the five routing profiles, their activating signals, and disambiguation rules. The classifier emits a profile $\pi \in \Pi_{\text{route}}$ and a self-reported confidence score $\rho \in [0, 1]$. When ρ falls below the router confidence threshold, or when the predicted profile violates a hard rule such as selecting precise without an exact lexical signal, the router falls back to semantic. Routing decisions are cached in a fixed-capacity least recently used (LRU) cache keyed by a SHA-256 prefix of the normalized query, reducing the cost of repeated or near-duplicate queries.

5.3.2 Multi-Path Retrieval with Weighted RRF

Each routing or recovery profile specifies a set of active retrieval paths P_π. For single-path profiles, weighted RRF effectively preserves the original path ranking. For multi-path profiles, especially the full recovery profile, ranked outputs from different paths are fused using weighted RRF. For a chunk d appearing at rank $r_p(d)$ in path p, the fused score is:

$$S(d) = \sum_{p \in P_\pi} w_\pi(p) \cdot \frac{1}{k + r_p(d)} \tag{5.12}$$

where $k = 60$ is the RRF stability constant and $w_\pi(p)$ is the profile-specific path weight. The fused list is filtered by a minimum RRF score threshold when configured and then truncated to a profile-specific candidate cap. When reranking is enabled, candidates are reranked by the cross-encoder bge-reranker-v2-m3 and filtered by a relevance threshold τ_{rerank}, yielding the final chunk set K.

5.3.3 Sufficiency Assessment

Before generation, a grader assesses whether the retrieved chunk set K contains enough evidence to answer the query q. The grader is implemented as a single VLM call. Its prompt

consists of a shared context prefix followed by a task-specific instruction. The shared prefix serializes retrieved chunks as numbered sources:

[1] Source: file path chunk text, [2] Source: file path chunk text, . . .

The grader emits a binary sufficiency judgment $\sigma \in \{0, 1\}$, an unanswerability signal $\upsilon \in \{0, 1\}$, and a one-sentence diagnostic rationale δ.

If $\sigma = 1$, the controller proceeds to generation. Otherwise, it enters the iterative refinement regime. The unanswerability signal υ is set only when the retrieved chunks explicitly state or clearly imply that the requested information is absent from the indexed documents. Empty or off-topic chunks are treated as retrieval failures rather than categorical unanswerability. When $\upsilon = 1$, the controller skips further retrieval and transitions directly to $\text{END}_{\text{insufficient}}$.

The grader, generator, and hallucination verifier use the same context-prefix construction. Under vLLM's automatic prefix caching, this allows the prefill computation for the chunk corpus to be reused across successive prompted calls when the retrieved context remains unchanged.

5.3.4 Iterative Refinement and Failure-Driven Escalation

When the grader returns $\sigma = 0$ and $\upsilon = 0$, the controller selects a deterministic remediation strategy according to the retrieval cycle c_r.

Cycle 0 — Query rewriting. At $c_r = 0$, a rewriter is invoked using the active query and the grader's diagnostic rationale δ. The rewriter targets surface-level retrieval mismatches, including ambiguous terminology, missing domain context, and compound noun phrases. The rewritten query replaces the active query, and retrieval is repeated under the same routed profile π. If rewriting fails or returns an empty result, the original query is retained.

Cycle 1 — Query decomposition. At $c_r = 1$, the controller invokes a decomposer on the original query. The decomposer produces at most four independent sub-questions, each self-contained and non-overlapping in scope. Each sub-question is retrieved under the FULL recovery profile to maximise recall. The resulting chunks are merged, deduplicated, and capped to prevent unbounded context growth. This is the terminal high-recall retrieval attempt in the `agentic` controller. If the subsequent grader call still finds the evidence insufficient, the controller abstains.

5.3.5 Grounded Generation and Hallucination Verification

When sufficiency is established, the generator produces a candidate answer a conditioned on K and q. The generation prompt instructs the model to answer only from the supplied

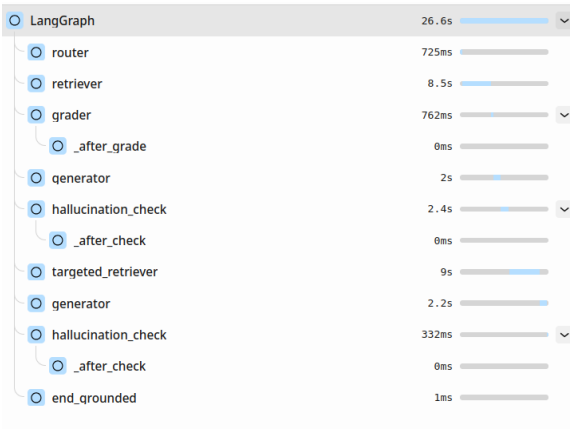


Figure 5.4: Execution trace of a single query through the agentic controller, captured via OpenTelemetry spans and visualised in the Arize Phoenix tracing UI (Arize AI, 2024). The span hierarchy directly reflects the finite-state machine of Figure 5.3.

context and to state explicitly when the context lacks the information needed to answer accurately.

The verifier then checks a against K. It emits a binary grounding judgment $\gamma \in \{0, 1\}$ and a set of unsupported claims $U = \{u_1, \ldots, u_\ell\}$. Binary grounding is used deliberately: in industrial document settings, even a single fabricated factual claim can propagate into downstream decisions with high cost.

When $\gamma = 0$, the controller performs targeted retrieval as long as the verification recovery budget is not exhausted. The targeted query is formed from the unsupported claims U, and retrieval is repeated under the original routed profile. Newly retrieved chunks are merged with the existing context, deduplicated, and capped. Generation and verification are then repeated on the enlarged context.

5.3.6 Termination

The controller has two terminal states. $\text{END}_{\text{grounded}}$ is reached when the hallucination verifier marks the generated answer as grounded; the answer is returned together with its source chunks. $\text{END}_{\text{insufficient}}$ is reached when the retrieval loop or the verification loop exhausts its budget without producing a grounded answer; the candidate answer is suppressed rather than returned as an unverified response, since in industrial settings a single fabricated claim can propagate into downstream decisions at high cost.

Both loops have hard upper bounds. The retrieval loop performs at most three sufficiency-grading rounds—initial routed retrieval, a rewritten-query retry, and a decomposed FULL-profile recovery—and the verification loop performs at most three answer-generation and verification attempts. The early unanswerable path short-circuits the escalation entirely after the first sufficiency judgment, costing only the router and grader calls (two invocations total); this avoids redundant retrieval cycles when the grader has identified a categorical knowledge gap, while retaining the full escalation path for queries that are difficult to retrieve but not categorically absent from the indexed corpus.

Figure 5.4 shows a single query executed end-to-end through the controller, captured

as OpenTelemetry spans. The trace illustrates the verification recovery path: an initial retrieval round produces chunks that the grader judges insufficient, the generator nevertheless emits a candidate answer, the hallucination check flags it as ungrounded, and a single `targeted_retriever` call followed by re-generation and re-verification recovers a grounded answer within the verification budget. The span tree mirrors the finite-state machine of Figure 5.3, providing a node-by-node view of which controller transitions fired on this input.

This chapter has addressed the two retrieval-layer limitations identified in Section 5.1. The chunk-augmented PPR traversal of Section 5.2 replaces LightRAG's one-hop neighbourhood expansion with multi-hop random-walk propagation. The agentic controller of Section 5.3, with its routed retrieval profiles, sufficiency grader, hallucination verifier, and bounded recovery loops, extends LightRAG's open-loop retrieval into a closed-loop adaptive workflow. Chapter 6 reports quantitative results and component ablations that isolate the contribution of each retrieval primitive.

Chapter 6
Evaluation

This chapter evaluates the proposed system from four perspectives. First, it measures end-to-end document QA performance on DocBench Academic. Second, it tests retrieval recall in a large scientific-literature setting using a controlled SurGE subset. Third, it evaluates multi-hop evidence discovery using 2WikiMultiHopQA, HotpotQA, and MuSiQue. Finally, it uses graph-level diagnostics to inspect the structure of the constructed knowledge graph.

Section 6.1 describes the experimental setup and baseline configurations. Section 6.2 describes the benchmark datasets and diagnostic tools used. Section 6.3 defines the evaluation metrics. Section 6.4 presents the main results. Section 6.5 summarizes the main findings.

6.1 Experimental Setup

This section describes the models, tools, baseline configurations, and retrieval settings used across the experiments.

6.1.1 Models

Document parsing tool

MinerU is used for document parsing before indexing. It converts raw document files into structured outputs through OCR and layout-aware parsing. The parsed outputs include text blocks, tables, formulas, figures, and layout-related information.

Embedding model

BGE-M3 is used to embed queries and retrieval units in the system (BAAI, 2024a; Chen et al., 2024). The embedded units include text chunks, textual representations of multimodal elements, and knowledge graph entities and relations generated during indexing. BGE-M3 is a multilingual model that supports dense, sparse, and multi-vector retrieval. It also accepts

inputs up to 8,192 tokens. These properties make it suitable for long-document retrieval in industrial settings.

Reranking model

Bge-reranker-v2-m3 is used to rerank retrieved candidates before generation (BAAI, 2024b; FlagOpen, 2024). It is a cross-encoder reranker that takes a query and a passage as joint input and outputs a relevance score. The score is used to reorder candidate chunks before constructing the final context.

Vision-language models (VLMs)

Three VLMs are evaluated as generators. They are used at different stages of system development and evaluation. All three are deployed locally via vLLM.

Qwen2-VL-7B-Instruct is used as the first lightweight VLM baseline (Qwen, 2024a; Wang et al., 2024b). It is used to test the VLM-dependent parts of the pipeline, including entity and relation extraction, multimodal element analysis, and answer generation from retrieved evidence. Because of its smaller scale, it provides a practical starting point for debugging these steps and establishing initial results.

InternVL2-26B-AWQ is used in the next stage of development (OpenGVLab, 2024a,b). It consists of a vision encoder, an MLP projector, and a language model backbone. We deploy the AWQ quantized checkpoint for efficient local inference. Compared with Qwen2-VL-7B-Instruct, InternVL2 provides a stronger multimodal generator while remaining feasible for local deployment. In our pipeline, InternVL2 is used for multimodal element analysis during indexing, entity and relation extraction for knowledge graph construction, and answer generation during querying.

Qwen3-VL-30B-A3B-Instruct-FP8 is used as the strongest VLM generator setting (Qwen, 2025; Bai et al., 2025). We run this model with FP8 quantization for local deployment. It is used in later experiments after the pipeline and retrieval components are stabilized. It serves as the main generator for the later-stage experiments reported in this thesis.

Judge model

Qwen2.5-32B-Instruct is used for DocBench evaluation (Qwen, 2024b; Yang et al., 2024). It is not part of the RAG pipeline. The official DocBench protocol uses GPT-4 as the judge. Due to local deployment constraints, we replace it with Qwen2.5-32B-Instruct. All DocBench results newly produced in this thesis use this judge. The published RAG-Anything result is included only as an external reference and follows its original evaluation setting.

Table 6.1 summarizes the model and tool stack.

6.1.2 Baseline Configurations

We define separate baselines for the standard KG retrieval path and the PPR retrieval path. The two paths use different candidate-selection mechanisms.

KG retrieval baseline

The KG retrieval baseline follows the standard RAG-Anything/LightRAG-style retrieval

Table 6.1: Models and tools used across the experiments.

Role	Model / Tool	Notes
Parser	MinerU	OCR and layout-aware document parsing
Embedding	BGE-M3	Embeddings for chunks, entities, and relations
Reranker	bge-reranker-v2-m3	Cross-encoder candidate reranking
VLM	Qwen2-VL-7B-Instruct	Lightweight VLM for early experiments
VLM	InternVL2-26B-AWQ	VLM for multimodal analysis and KG extraction
VLM	Qwen3-VL-30B-A3B-FP8	Main VLM in later-stage experiments
Judge	Qwen2.5-32B-Instruct	DocBench scoring only

pipeline. Given a query, the system first extracts low-level and high-level keywords. Keywords at the same level are joined into one retrieval query. Low-level keywords are used for entity retrieval. High-level keywords are used for relation retrieval.

The baseline uses dense vector retrieval over the entity and relation vector stores. It does not use BM25 or hybrid retrieval. After entities and relations are retrieved, textual KG evidence is token-truncated. Source chunks are collected from the retained entity and relation evidence. The resulting chunk candidates are reranked by the cross-encoder reranker. The final prompt uses the KG-message format, which includes entity evidence, relation evidence, and retrieved chunk content.

Thus, the KG retrieval baseline uses joined keywords, dense retrieval, truncated KG evidence, chunk reranking, and KG-message prompting.

PPR retrieval baseline

The PPR baseline uses the same indexed graph but replaces the standard KG chunk-selection path with graph propagation. It starts from query-derived graph seeds and applies PPR to rank graph-based chunk candidates. The baseline PPR setting uses dense seed retrieval and PPR-ranked chunk selection. It does not use synonym edges, hybrid retrieval, per-keyword retrieval, downstream chunk reranking, or RRF fusion unless explicitly stated.

Thus, the PPR baseline uses joined keywords, dense seed retrieval, PPR graph propagation, and PPR-ranked chunk selection.

6.1.3 Retrieval Component Switches

Table 6.2 defines the retrieval component switches used in the experiments. Rows prefixed with "+" denote single-component additions relative to the corresponding KG or PPR baseline. Rows prefixed with "w/o" denote removal-style ablations from a full configuration.

6.1.4 Retrieval Configuration

We keep the main retrieval settings fixed within each experiment group. `top_k` controls the number of retrieved entities and relations. For standard KG retrieval, the number of chunks kept after retrieval is controlled by `chunk_top_k`. For PPR retrieval, the graph propagation

Table 6.2: Retrieval component switches evaluated in the experiments.

Component	Definition
KG rerank	Reranks retrieved entities and relations before token truncation.
Per-keyword retrieval	Retrieves each extracted keyword separately instead of joining same-level keywords.
Hybrid retrieval	Replaces dense-only retrieval with dense + BM25 retrieval.
Untruncated KG evidence	Collects source chunks from all retrieved entity/relation evidence before token truncation.
Chunk-only message	Uses only retrieved chunks in the final prompt.
Synonym edges	Adds similarity-based edges between semantically similar entity nodes without merging nodes. The cosine threshold is set to 0.80.
PPR chunk rerank	Reranks PPR-selected chunks with the cross-encoder reranker.
RRF	Fuses multiple ranked lists using RRF.

step first ranks graph-related chunks, and `ppr_qa_top_k` controls how many of these chunks are kept.

For DocBench, we use `top_k`=20 and keep 10 chunks for the final answer context. We also set multimodal_top_k=3 to limit the number of multimodal items passed to the VLM. For SurGE, we use `top_k`=20 and keep 50 chunks for query-level retrieval and 500 chunks for survey-level retrieval. These values match the largest Recall@k reported for each SurGE setting. For Multi-hop QA, we use `top_k`=10 and keep the top 5 chunks unless otherwise stated.

For standard KG retrieval, entity and relation evidence are constrained by `max_entity_tokens`=6000 and `max_relation_tokens`=8000. The truncated and untruncated settings differ in whether source chunks are collected after or before this token truncation. Full retrieval settings are reported in Appendix C.

6.2 Datasets

6.2.1 DocBench

DocBench is a benchmark for evaluating LLM-based document reading systems (Zou et al., 2024). It covers multiple domains and includes both answerable and unanswerable questions.

Subset selection. We evaluate on the *Academic* domain only. This subset matches our target use case and reduces evaluation cost. The Academic domain contains 49 documents (Doc0–Doc48) and 303 questions.

Question labels. DocBench labels each question by the primary evidence source required: (i) *text-only*, answerable from plain text; (ii) *multimodal-t*, requiring tables; (iii) *multimodal-f*, requiring figures or diagrams; (iv) *meta-data*, requiring document-level metadata; (v) *unanswerable*, not supported by document evidence; and (vi) *una-web*, requiring external knowledge.

6.2.2 SurGE

SurGE is a benchmark for scientific survey generation in computer science (Su et al., 2026). We use it to evaluate retrieval recall in a large-scale scientific literature setting. SurGE provides 205 ground-truth surveys and a corpus of 1,086,992 arXiv papers. The released corpus is metadata-based. Each paper entry contains fields such as title, authors, abstract, category, and `doc_id`.

Subset construction. The full corpus is too large for iterative graph-based indexing. We therefore construct a controlled subset for retrieval evaluation. The resulting subset is released through Hugging Face for reproducibility (Cai and Zu, 2026). We select high-quality queries with `cite_extract_rate`=1.0, anchor a small set of ground-truth surveys, and collect their cited papers as target documents. Additional non-relevant papers are added as distractors to form a local retrieval corpus.

Chunk construction. Since SurGE provides abstracts rather than full paper text, each paper abstract is used as one chunk. This keeps a one-to-one mapping between chunks and doc_ids, which is important for citation-level recall evaluation. The final subset contains 12,156 abstract chunks, 59 query-level retrieval examples, and 10 survey-level examples.

6.2.3 Multi-hop QA Benchmarks

We also evaluate the retrieval pipeline on three multi-hop QA benchmarks: 2WikiMultiHopQA, HotpotQA, and MuSiQue. These datasets require a system to retrieve and combine evidence from multiple supporting passages before answering a question. They are therefore useful for testing multi-hop evidence discovery.

2WikiMultiHopQA is constructed from Wikipedia-style structured and unstructured knowledge (Ho et al., 2020). It provides questions that require reasoning over multiple supporting facts. HotpotQA contains natural multi-hop questions over Wikipedia and provides supporting facts for explainable question answering (Yang et al., 2018). MuSiQue builds multi-hop questions by composing connected single-hop questions (Trivedi et al., 2022). This design reduces shortcut reasoning and makes evidence composition more important.

6.2.4 GraphRAG-Bench

We use GraphRAG-Bench to diagnose the structural quality of the constructed knowledge graph (GraphRAG-Bench, 2025). We apply its indexing evaluation script to the GraphML files produced by our pipeline. The goal is to inspect graph-scale and connectivity properties, rather than QA accuracy. These diagnostics help compare graph construction variants and reveal structural issues that may affect downstream retrieval performance. The key metrics are described in Section 6.3.

6.3 Evaluation Metrics

DocBench accuracy. DocBench uses an LLM-as-a-judge protocol. The judge compares a system answer with the reference answer and assigns a binary score. The official setup uses

GPT-4 as the judge. We replace it with Qwen2.5-32B-Instruct due to deployment constraints. Accuracy is the fraction of questions judged correct.

SurGE retrieval recall. We report Recall@k, defined as the fraction of ground-truth cited documents recovered in the top-k retrieved results. Query-level evaluation uses small citation sets and reports Recall@5 through Recall@50. Survey-level evaluation uses larger citation sets and reports Recall@50 through Recall@500.

Multi-hop QA metrics. For the multi-hop QA benchmarks, we report Exact Match (EM), token-level F1, and retrieval recall. EM measures whether the predicted answer exactly matches the reference answer after normalization. F1 measures token overlap between the predicted answer and the reference answer. Recall@k measures whether the gold supporting evidence is retrieved within the top-k results. We report Recall@2, Recall@5, and Recall@10 depending on the experiment setting.

GraphRAG-Bench indexing metrics. GraphRAG-Bench indexing metrics are used to evaluate the structural quality of the constructed knowledge graph. Node and edge counts describe graph scale. Average degree $\bar{d} = 2|E|/|V|$ measures connectivity density. Isolated node count and connected component count measure graph fragmentation. The largest connected component indicates how much of the graph is covered by the main connected structure. The average clustering coefficient measures local community structure:

$$C_v = \frac{2e_v}{k_v(k_v - 1)}, \quad k_v \geq 2 \tag{6.1}$$

where k_v is the degree of node v and e_v is the number of edges among its neighbors. In this thesis, these metrics are used to compare graph construction variants, including the shared DocBench graph before and after adding synonym edges.

6.4 Results

This section reports the main experimental results. The results are organized by dataset and evaluation purpose. DocBench is used for end-to-end document question answering. SurGE is used for retrieval recall in scientific literature. The multi-hop QA benchmarks are used to evaluate graph-based evidence discovery and multi-hop answering. GraphRAG-Bench is used as a structural diagnostic for the constructed knowledge graph.

6.4.1 End-to-End DocBench Results

Iterative System Development

DocBench is evaluated in a per-document setting. Each question is associated with a target document, and retrieval is performed within that document. Thus, each document should have its own isolated index and knowledge graph during evaluation. This follows the official DocBench-style document reading setup.

Table 6.3 reports DocBench Academic accuracy across the main system development stages. The early rows marked as "shared kv" were development runs with an incorrect storage setup. Although the evaluation was intended to be per-document, the kv_store was shared across documents. This was later identified as a storage isolation issue. During retrieval, the

Table 6.3: DocBench Academic accuracy (%) across system development stages in the per-document setting. mm-t: multimodal-table; mm-f: multimodal-figure; unans: unanswerable; web: una-web. The published RAG-Anything result is included as an external reference.

System stage	Overall	text	mm-t	meta	mm-f	unans	web
Qwen2-VL (shared kv)	21.12	24.59	26.09	6.25	35.71	3.33	0.00
InternVL2 (shared kv)	44.22	45.90	56.52	14.58	66.67	20.00	0.00
InternVL2	52.81	63.93	59.13	25.00	71.43	36.67	0.00
Qwen3-VL	67.99	77.05	79.13	35.42	80.95	56.67	0.00
Qwen3-VL (final per-doc)	**71.62**	83.61	83.48	41.67	85.71	46.67	0.00
RAG-Anything	61.40	—	—	—	—	—	—

system could access chunks, entities, or relations from documents other than the current target document. These early results are therefore kept as development references rather than final comparable results.

After fixing the storage isolation issue, InternVL2 reaches 52.81% overall accuracy. The later switch to Qwen3-VL improves the overall accuracy to 67.99%. The final per-document configuration reaches 71.62%. This is above the reported RAG-Anything reference value of 61.40%, although the comparison is not strictly controlled because the generator and evaluation setup differ.

Shared-Graph Retrieval Components

The shared-graph setting is different from the storage bug in the early development runs. All 49 Academic documents are intentionally indexed into a single knowledge graph. This setting is not the official per-document DocBench setup. Instead, it simulates a more realistic retrieval scenario where the system must search across a document collection. It also allows us to evaluate whether graph-based retrieval components remain effective when cross-document evidence and distractors are present.

Tables 6.4 and 6.5 report component-level results under the shared-graph DocBench setting. Each non-baseline row changes the corresponding retrieval-path baseline independently; the rows are not cumulative. In the PPR table, *Chunk rerank + RRF* is the only two-component variant and is included to test whether rank fusion further improves post-PPR chunk reranking.

For the KG retrieval path, hybrid retrieval gives the best overall accuracy, improving the baseline from 53.80% to 58.75%. KG reranking and the chunk-only message format also improve the overall score, suggesting that both candidate filtering and final context construction affect answer quality in the shared-graph setting. Per-keyword retrieval gives only a small overall gain, while untruncated KG evidence does not improve the baseline. This suggests that expanding source-chunk collection before KG evidence truncation is not sufficient when the graph contains cross-document distractors.

For the PPR path, hybrid retrieval gives the best overall result, improving the baseline from 54.79% to 60.07%. Chunk reranking gives the best multimodal-figure result. Synonym edges and per-keyword retrieval bring small and category-specific gains, while RRF fusion

Table 6.4: DocBench Academic accuracy for independent single-component additions to the KG baseline in the shared-graph setting. Rows are not cumulative. Best values in each column are shown in bold.

Added component	Overall	text	mm-t	meta	mm-f	unans	web
None (KG baseline)	53.80	75.41	66.09	4.17	80.95	16.67	0.00
Chunk-only message	56.44	75.41	68.70	**6.25**	80.95	**30.00**	0.00
KG rerank	56.77	75.41	71.30	**6.25**	78.57	26.67	0.00
Per-keyword retrieval	54.46	75.41	67.83	4.17	**83.33**	13.33	0.00
Hybrid retrieval	**58.75**	**83.61**	**73.04**	**6.25**	80.95	20.00	0.00
Untruncated KG evidence	53.80	70.49	68.70	2.08	80.95	20.00	0.00

Table 6.5: DocBench Academic accuracy for independent component additions to the PPR baseline in the shared-graph setting. Rows are not cumulative; *Chunk rerank + RRF* is a two-component variant. Best values in each column are shown in bold.

Added component(s)	Overall	text	mm-t	meta	mm-f	unans	web
None (PPR baseline)	54.79	70.49	71.30	4.17	73.81	26.67	0.00
Synonym edges	55.78	67.21	72.17	6.25	78.57	30.00	0.00
Hybrid retrieval	**60.07**	**73.77**	**77.39**	**10.42**	76.19	**33.33**	**14.29**
Per-keyword retrieval	55.78	68.85	70.43	8.33	76.19	**33.33**	0.00
Chunk rerank	59.74	**73.77**	76.52	**10.42**	**83.33**	26.67	0.00
Chunk rerank + RRF	57.76	**73.77**	74.78	8.33	78.57	23.33	0.00

improves over the PPR baseline but remains below direct chunk reranking overall. These results show that retrieval design is important in the shared-graph setting, where cross-document distractors make candidate selection more difficult. Additional removal-style component sensitivity results are reported in Appendix D.

6.4.2 Retrieval Recall on SurGE

SurGE is used to evaluate retrieval recall in a scientific-literature setting. The query-level setting evaluates short citation-oriented queries. The survey-level setting is harder because each survey is associated with a larger citation set. We report KG retrieval and PPR retrieval in the same table for each setting. Within each retrieval path, every non-baseline row changes the corresponding baseline independently; the rows are not cumulative.

Query-Level Recall

Table 6.6 reports SurGE query-level Recall@k. In the KG path, hybrid retrieval gives the best recall across all reported cutoffs. KG reranking and untruncated KG evidence do not improve query-level recall, and per-keyword retrieval reduces recall at larger cutoffs. In the PPR path, synonym edges give only a small gain over the baseline, while per-keyword retrieval does not

Table 6.6: SurGE query-level Recall@*k* for independent component additions to the KG and PPR baselines. Rows are not cumulative; *Chunk rerank + RRF* is a two-component variant. Best values in each column are shown in bold.

Path	Added component(s)	R@5	R@10	R@20	R@30	R@50
KG	None (baseline)	0.065	0.120	0.188	0.236	0.324
KG	KG rerank	0.065	0.116	0.188	0.240	0.305
KG	Per-keyword retrieval	0.068	0.109	0.163	0.195	0.263
KG	Hybrid retrieval	**0.078**	**0.125**	0.190	0.242	0.333
KG	Untruncated KG evidence	0.065	0.120	0.183	0.233	0.317
PPR	None (baseline)	0.054	0.093	0.121	0.170	0.293
PPR	Synonym edges	0.058	0.091	0.121	0.190	0.297
PPR	Hybrid retrieval	0.044	0.098	0.140	0.221	0.329
PPR	Per-keyword retrieval	0.060	0.073	0.123	0.200	0.300
PPR	Chunk rerank	**0.078**	**0.125**	**0.218**	**0.302**	**0.410**
PPR	Chunk rerank + RRF	0.065	0.110	0.215	0.274	0.381

improve recall consistently. Hybrid retrieval improves recall from R@10 onward but lowers R@5. Adding chunk reranking gives the best result at every cutoff. RRF fusion improves over the PPR baseline but remains below direct chunk reranking. The best query-level R@50 is 0.410.

Survey-Level Recall

Table 6.7 reports SurGE survey-level Recall@*k*. Within the KG path, untruncated entity/relation evidence gives the best recall at all reported cutoffs. This suggests that survey-level citation recall benefits from broader KG evidence before token truncation. However, KG reranking, per-keyword retrieval, and hybrid retrieval do not improve over the KG baseline in this setting. At larger retrieval depths, PPR-based methods give much higher recall. Within the PPR path, synonym edges provide a modest gain across all reported cutoffs. Hybrid retrieval helps at intermediate cutoffs but not at R@500, while per-keyword retrieval consistently reduces recall in this setting. PPR with chunk reranking gives the best R@500, and RRF fusion performs almost identically but does not exceed direct chunk reranking.

6.4.3 Multi-hop QA Results

The multi-hop QA experiments evaluate whether graph-based retrieval improves answer generation when a question requires evidence from multiple passages. We report Exact Match (EM), token-level F1, and retrieval recall.

We organize the multi-hop results in two stages. First, we report a non-CoT baseline comparison under the original RAG answer prompt. This comparison focuses on the effect of replacing the standard KG retrieval path with PPR retrieval. Second, we report the final multi-hop setting, which uses a CoT-style answer prompt and focuses on PPR-based retrieval components. The CoT-style prompt is used because multi-hop QA requires the model to

Table 6.7: SurGE survey-level Recall@*k* for independent component additions to the KG and PPR baselines. Rows are not cumulative; *Chunk rerank + RRF* is a two-component variant. Best values in each column are shown in bold.

Path	Added component(s)	R@50	R@100	R@200	R@500
KG	None (baseline)	0.168	0.242	0.246	0.246
KG	KG rerank	0.165	0.221	0.223	0.223
KG	Per-keyword retrieval	0.151	0.216	0.232	0.232
KG	Hybrid retrieval	0.157	0.206	0.208	0.208
KG	Untruncated KG evidence	**0.169**	0.264	0.281	0.281
PPR	None (baseline)	0.122	0.217	0.351	0.561
PPR	Synonym edges	0.128	0.247	0.379	0.574
PPR	Hybrid retrieval	0.125	0.235	0.363	0.559
PPR	Per-keyword retrieval	0.106	0.195	0.321	0.554
PPR	Chunk rerank	0.163	**0.285**	**0.424**	**0.596**
PPR	Chunk rerank + RRF	0.163	**0.285**	**0.424**	0.595

Table 6.8: Multi-hop QA results on 2WikiMultiHopQA, HotpotQA, and MuSiQue. Best values in each column are shown in bold.

Method	2WikiMultiHopQA				HotpotQA				MuSiQue			
	EM	F1	R@2	R@5	EM	F1	R@2	R@5	EM	F1	R@2	R@5
Naive	0.3700	0.4363	0.6480	0.6647	0.4540	0.5970	**0.8015**	0.8570	0.1160	0.1873	0.3212	0.3591
KG baseline	0.3990	0.4872	0.6168	0.6320	0.4600	0.5986	0.7680	0.8235	0.1530	0.2534	0.3008	0.3367
PPR baseline	**0.4950**	**0.5958**	**0.7312**	**0.9125**	**0.4820**	**0.6236**	0.6775	**0.8650**	**0.1730**	**0.2788**	**0.3826**	**0.5210**

combine evidence across multiple retrieved passages.

Earlier component and ablation runs under the original RAG answer prompt are reported in Appendix E. These runs are kept as exploratory results because they reuse component settings originally developed for DocBench and are less representative of the final multi-hop configuration.

PPR versus Baselines

Table 6.8 compares naive retrieval, the KG retrieval baseline, and PPR retrieval under the standard non-CoT RAG answer prompt. The prompt asks the model to answer concisely from the retrieved context, without explicit chain-of-thought reasoning instructions. This table provides the non-CoT baseline comparison for the later multi-hop experiments. PPR gives the strongest answering performance on all three datasets in terms of EM and F1. It also improves R@5 on all three datasets. These results motivate the later focus on PPR-based retrieval for multi-hop QA.

Table 6.9: Multi-hop QA results for independent single-component additions to the PPR baseline under the CoT-style prompt. Rows are not cumulative. Best values in each column are shown in bold.

Added component	2WikiMultiHopQA				HotpotQA				MuSiQue			
	EM	F1	R@2	R@5	EM	F1	R@2	R@5	EM	F1	R@2	R@5
None (PPR baseline)	0.600	0.6917	0.7365	0.9163	0.541	0.6788	0.6855	0.8630	0.281	0.3933	0.3872	0.5313
Hybrid retrieval	**0.617**	**0.7097**	**0.7555**	**0.9327**	**0.557**	**0.6968**	**0.7125**	**0.8945**	**0.287**	**0.3991**	**0.3989**	**0.5443**
Per-keyword retrieval	0.610	0.6939	0.7368	0.9113	0.525	0.6665	0.6670	0.8515	0.251	0.3663	0.3907	0.5343
Synonym edges	0.587	0.6829	0.7360	0.9143	0.544	0.6823	0.6875	0.8655	0.266	0.3759	0.3854	0.5246

Table 6.10: Multi-hop QA results for adding per-keyword retrieval and synonym edges on top of PPR hybrid retrieval under the CoT prompt. Best values in each column are shown in bold.

Added component(s)	2WikiMultiHopQA				HotpotQA				MuSiQue			
	EM	F1	R@2	R@5	EM	F1	R@2	R@5	EM	F1	R@2	R@5
Per-keyword	0.593	0.6847	0.7558	0.9230	0.540	0.6896	0.6910	0.8795	**0.284**	0.3902	**0.4112**	0.5428
Synonym edges	**0.629**	**0.7229**	**0.7592**	**0.9340**	**0.564**	**0.7068**	**0.7150**	**0.8995**	0.283	**0.3994**	0.4024	**0.5533**
Per-keyword + Synonym	0.613	0.7012	0.7578	0.9255	0.548	0.6892	0.6935	0.8860	0.282	0.3935	0.4092	0.5404

PPR Component Results with CoT Prompt

Table 6.9 reports PPR component results after applying the CoT-style prompt. The CoT-style prompt asks the model to first identify the relevant evidence and then produce a concise final answer. In our multi-hop experiments, this provides a stronger answer-generation setting than the earlier non-CoT RAG prompt. The table evaluates three single-component additions under this setting: hybrid retrieval, per-keyword retrieval, and synonym edges.

Hybrid retrieval gives the most consistent improvement. It improves EM, F1, R@2, and R@5 on all three datasets. Per-keyword retrieval and synonym edges do not provide consistent gains when used alone.

Hybrid and Synonym Combination Results

Table 6.10 further evaluates combinations built on top of hybrid retrieval. Adding synonym edges to hybrid retrieval gives the best EM and F1 on 2WikiMultiHopQA and HotpotQA. It also gives the best F1 and R@5 on MuSiQue. Adding both per-keyword retrieval and synonym edges does not further improve the result. This suggests that synonym edges are useful in the hybrid PPR setting, while per-keyword retrieval may introduce additional noise when combined with both hybrid retrieval and synonym expansion. Additional synonym-threshold sensitivity and Optuna-based PPR hyperparameter optimization results are reported in Appendix F.

Table 6.11: Agentic retrieval results on 2WikiMultiHopQA, HotpotQA, and MuSiQue. Best values among reported values in each column are shown in bold.

Method	2WikiMultiHopQA					HotpotQA					MuSiQue				
	EM	F1	R@2	R@5	R@10	EM	F1	R@2	R@5	R@10	EM	F1	R@2	R@5	R@10
Chunk $k = 5$	0.622	0.7117	**0.7465**	**0.9225**	—	0.589	0.7245	**0.7665**	**0.9090**	—	0.283	0.4015	**0.3929**	**0.5498**	—
Chunk $k = 10$	**0.634**	**0.7239**	0.7412	0.8865	0.9527	**0.593**	**0.7325**	0.7640	0.8895	0.9375	**0.312**	**0.4269**	0.3812	0.5069	0.6129

Agentic Retrieval

Table 6.11 reports the agentic retrieval results. The setting with chunk $k = 10$ gives the best EM and F1 on all three datasets. The chunk $k = 5$ setting gives higher R@2 and R@5, while chunk $k = 10$ provides broader evidence coverage. This suggests a trade-off between retrieving a compact set of highly ranked evidence and providing a larger context for answer generation.

6.4.4 GraphRAG-Bench Structural Diagnostics

We use GraphRAG-Bench as a structural diagnostic for the constructed knowledge graph. These metrics are not used as final task scores. Instead, they are used to inspect graph scale, connectivity, fragmentation, and local clustering. Table 6.12 reports the structural statistics of the shared DocBench graph before and after adding synonym edges. The synonym-edge graph uses the default cosine threshold of 0.80.

Adding synonym edges does not change the number of nodes, since no entity nodes are merged or removed. It increases the number of edges from 51,795 to 74,904 and raises the average degree from 5.5399 to 8.0116. The number of isolated nodes drops from 772 to 363, and the largest connected component grows from 17,556 to 17,862 nodes. The average clustering coefficient also increases from 0.5953 to 0.6100. These changes show that synonym edges improve local connectivity and reduce the number of isolated nodes while keeping the entity set fixed. The number of non-isolated components increases from 119 to 155, suggesting that some previously isolated nodes are connected into small components rather than being fully absorbed into the largest component. Thus, synonym edges reduce isolation and strengthen connectivity, but they do not fully solve graph fragmentation.

6.5 Evaluation Summary

The evaluation leads to five main findings. First, the final per-document DocBench configuration reaches 71.62% accuracy. This is higher than the reported RAG-Anything reference baseline, although the comparison is not strictly controlled because the generator and evaluation setup differ.

Second, in the shared-graph DocBench setting, retrieval design has a clear effect on accuracy. Hybrid retrieval gives the strongest overall result for both KG and PPR paths, while other component switches mainly give smaller or category-specific gains.

Table 6.12: GraphRAG-Bench structural diagnostics for the shared DocBench graph before and after adding synonym edges.

Metric	Without synonym edges	With synonym edges
Nodes	18,699	18,699
Edges	51,795	74,904
Average degree	5.5399	8.0116
Connected components	891	518
Non-isolated components	119	155
Largest component size	17,556	17,862
Isolated nodes	772	363
Non-isolated nodes	17,927	18,336
Average clustering coefficient	0.5953	0.6100

Third, on SurGE, PPR with chunk reranking gives the best retrieval recall among the tested settings. This holds for both query-level retrieval and survey-level retrieval. Within the KG path, untruncated entity/relation evidence gives the best survey-level recall, showing that broader KG evidence can help when the target citation set is large.

Fourth, on the multi-hop QA benchmarks, PPR-based retrieval improves answer quality over the KG and naive baselines under the non-CoT prompt. Under the CoT-style prompt, hybrid retrieval is the most consistent single PPR component. Among non-agentic PPR combinations, hybrid retrieval with synonym edges gives the strongest overall multi-hop configuration. The agentic retrieval setting with chunk $k = 10$ gives the strongest EM and F1 among all reported multi-hop settings. Within the two agentic settings, chunk $k = 5$ gives higher R@2 and R@5.

Fifth, GraphRAG-Bench diagnostics show that synonym edges improve graph connectivity and reduce isolated nodes. However, they do not fully remove graph fragmentation.

Overall, the results suggest that graph propagation and hybrid retrieval provide stable gains across the main QA settings. Agentic control further improves multi-hop answer metrics, but its benefit depends on the chunk budget and retrieval setting. These patterns are analyzed in detail in Chapter 7.

Chapter 7
Discussion

This chapter interprets the experimental results reported in Chapter 6, examines limitations of the proposed system, and outlines directions for future work. We interpret the main findings by benchmark and highlight patterns that recur across experiments.

7.1 Result Patterns Across Benchmarks

The experiments reveal different behaviours across the four evaluation settings. DocBench per-document evaluation mainly tests single-document multimodal question answering. The shared-graph DocBench setting tests retrieval under cross-document distractors. SurGE evaluates large-scale citation-oriented recall. The multi-hop QA benchmarks test whether graph traversal helps recover evidence chains needed for answer generation.

Per-document DocBench. The DocBench Academic results in Table 6.3 show that the best configuration performs strongly on the *multimodal-figure* and *multimodal-table* categories, reaching 85.71% and 83.48%, respectively. We attribute this to the interaction between multimodal indexing and graph-based retrieval.

First, VLM-generated descriptions of figures, tables, and formulas are inserted into the retrieval and graph-construction pipeline rather than attached only as weak captions. Second, graph-based retrieval can connect textual evidence with multimodal element descriptions through shared entities and relations. When PPR is enabled, this connectivity further allows query relevance to propagate beyond direct lexical or embedding matches. The combined effect is that questions whose evidence is anchored in a non-textual element can still be answered through a chain that begins in text.

Shared-graph DocBench. The shared-graph DocBench setting provides a retrieval stress test rather than a replacement for the official per-document setup. In this setting, all Academic documents are indexed into a single graph, so retrieval must select evidence from

a larger pool with cross-document distractors. This setting also changes the nature of some DocBench questions. DocBench questions are originally written with a fixed target document in mind, so they often use document-internal references, such as figure numbers, author order, or phrases like "this paper", without explicitly naming the document. This is natural in the per-document setting, where the target document is already known. In the shared-graph setting, however, such queries become under-specified for retrieval because the system must first identify the intended document before selecting evidence. This partly explains why the shared-graph results are substantially lower than the final per-document DocBench result and should not be treated as a direct replacement for the official setup.

Within this harder shared-graph setting, the component results still show clear retrieval effects. Hybrid retrieval gives the strongest overall result for both the KG path and the PPR path. This indicates that combining lexical and semantic matching is useful when retrieval must operate over a multi-document graph with many distractors.

SurGE retrieval. SurGE separates short query-level retrieval from broader survey-level retrieval. At the query level, improvements are modest, with the best R@50 reaching 0.410. At the survey level, PPR becomes more useful at larger retrieval depths, reaching the best R@500 when combined with chunk reranking. This pattern suggests that, on SurGE, PPR is more useful as a high-depth expansion mechanism than as a top-rank precision mechanism.

However, SurGE differs from multi-hop QA. It does not mainly require a tightly chained reasoning path across two supporting passages. Instead, it requires broad citation-oriented semantic coverage. The absolute recall therefore remains limited for downstream survey generation. The likely reason is a mismatch between the retrieval signal and the evaluation target. Our graph is built over arXiv abstracts, so it mainly captures entity co-occurrence and local semantic similarity in short metadata fields. In contrast, survey citations often reflect broader scholarly relevance and topic coverage, which are not always explicit in abstracts.

Multi-hop QA and agentic retrieval. The multi-hop QA results show that PPR retrieval improves answer quality over the KG and naive baselines in the non-CoT setting. After switching to the CoT-style answer prompt, hybrid retrieval becomes the most consistent single component, improving EM, F1, R@2, and R@5 on all three datasets. This suggests that PPR provides graph-based evidence propagation, while hybrid retrieval supplies the lexical–semantic anchoring needed to construct reliable seeds and retrieve directly matching passages.

The strongest non-agentic PPR setting combines hybrid retrieval with synonym edges. This setting indicates that synonym edges are more useful when added to a strong hybrid retrieval signal than when used as a standalone component. The agentic results should therefore be interpreted against this strongest non-agentic setting. With chunk $k = 5$, agentic retrieval is slightly worse on 2WikiMultiHopQA, better on HotpotQA, and nearly tied on MuSiQue. With chunk $k = 10$, agentic retrieval gives the best EM and F1 on all three datasets, but the gains over the strongest non-agentic setting are modest. Early-rank recall does not improve uniformly: chunk $k = 5$ gives higher R@2 and R@5 than chunk $k = 10$, while the strongest non-agentic setting remains competitive on R@2 and R@5.

This suggests that the current agentic pipeline mainly improves how retrieved evidence is selected and used for answer generation, rather than uniformly improving retrieval recall.

Since the agentic pipeline requires additional LLM calls for routing, grading, rewriting, or verification, its accuracy gains must be weighed against higher inference cost. These results leave substantial room for improving the controller policy and the calibration of its intermediate decisions.

Component transfer across tasks. The component results show that not all retrieval switches transfer equally across benchmarks. Hybrid retrieval is the most robust component: it gives the best overall result in the shared-graph DocBench setting for both KG and PPR paths, and it is also the most consistent single addition in the multi-hop CoT setting. This suggests that sparse lexical matching remains important even when dense retrieval and graph propagation are available.

Other components are more task-dependent. KG reranking improves DocBench shared-graph accuracy but does not improve SurGE recall. Per-keyword retrieval gives no consistent gain. Compared with the joined-keyword, retrieving each keyword separately weakens the joint semantic constraint and can introduce candidates that match only part of the query intent. Untruncated KG evidence is not helpful for DocBench QA, but it improves KG-path recall in the SurGE survey-level setting, where broad citation coverage is more important than compact answer evidence. Chunk-only prompting improves the KG path on DocBench, suggesting that raw entity and relation text can sometimes add noise to the final answer context. RRF fusion improves over the PPR baseline on SurGE, but it does not outperform direct chunk reranking. These patterns support using component switches as task-specific controls rather than as universally beneficial additions.

Chunk reranking across tasks. Chunk reranking behaves differently across tasks. In the standard KG path, chunk reranking is part of the default retrieval pipeline. The explicit switch evaluated in this thesis is therefore mainly post-PPR chunk reranking, where PPR-selected chunks are reranked by a cross-encoder before answer generation.

On DocBench and SurGE, this type of reranking is useful because the evaluation rewards direct relevance filtering. DocBench requires selecting evidence that directly supports a document question, while SurGE requires ranking citation-oriented candidates by semantic relevance. In the PPR path, chunk reranking gives the best SurGE query-level R@50 and the best survey-level R@500.

The pattern changes on multi-hop QA. In the earlier non-CoT PPR component runs reported in Appendix E, adding chunk reranking reduces answer quality on 2WikiMultiHopQA and MuSiQue. On HotpotQA, it improves R@2 and R@5 but still lowers EM and F1 compared with the PPR baseline. This suggests that chunk-wise reranking can surface passages that are individually similar to the query, but it may weaken the multi-hop evidence set needed for answer generation. Multi-hop QA often requires a bridge passage or a second-hop passage whose surface form is not directly similar to the original query. A cross-encoder that scores each chunk independently may therefore down-rank useful bridge evidence. For this reason, chunk reranking is not used as the default PPR setting in the final multi-hop QA experiments.

Synonym linking. The synonym-threshold sweep in Table F.1 shows that EM and F1 remain relatively stable across thresholds from 0.70 to 0.90, with most variations within roughly two to three percentage points. The component results further show that synonym

linking is not a major standalone source of accuracy improvement: adding synonym edges alone does not give stable gains across the multi-hop QA datasets, whereas adding them on top of hybrid PPR retrieval gives the strongest non-agentic configuration. This suggests that synonym edges are most useful when the retrieval seeds are already well anchored by hybrid lexical–semantic matching. Their role is to recover surface-form mismatches, such as abbreviations, transliterations, and cross-lingual variants, that would otherwise reduce recall on a subset of queries. The GraphRAG-Bench diagnostics in Chapter 6 support this interpretation at the graph level: synonym edges reduce isolated nodes and improve connectivity without changing the entity set.

7.2 Limitations

Baseline comparability. Our headline number on DocBench Academic compares a system built on Qwen3-VL-30B-A3B-FP8 with the published RAG-Anything reference, which uses a different generator. The same-model comparison in Table 6.3 is therefore the more faithful measure of the contribution of this thesis. It shows that the final per-document configuration improves accuracy from 67.99% to 71.62%. This 3.63-point gain is the portion most directly associated with the retrieval and indexing changes introduced in the final configuration. The larger gap to the published RAG-Anything reference should be interpreted cautiously, since it may also reflect generator choice, implementation details, and the judge-model substitution discussed below.

Judge-model substitution. The published RAG-Anything reference evaluates DocBench answers using GPT-4o-mini. In our local evaluation, we use Qwen2.5-32B-Instruct as the LLM judge with the same evaluation prompt and scoring format. We have not measured judge agreement on a held-out sample. Absolute accuracy figures should therefore be read as comparable across our internal runs, but not as strictly comparable to externally published DocBench numbers.

Statistical significance. Results in Chapter 6 are reported from a single decoding pass, and we do not provide confidence intervals or multi-seed estimates. We therefore avoid making strong claims from isolated small gaps, especially in component-level comparisons. On DocBench Academic, a three-point difference corresponds to about nine questions out of 303; on the multi-hop QA datasets, one EM point corresponds to about ten questions. Our main claims are based on directional patterns that recur across datasets, metrics, and component comparisons, rather than on individual small differences.

SurGE retrieval recall. The SurGE subset is the most demanding retrieval benchmark in our evaluation. Although PPR with chunk reranking improves high-depth recall, the absolute recall remains below the level needed for reliable downstream survey generation. We therefore report SurGE as a stress test that exposes the boundary of the proposed graph-based retrieval design at corpus scale. A stronger SurGE system would likely require signals that are not present in our current abstract-only graph, such as full-text evidence, citation links, venue and temporal metadata, or paper-level influence signals.

Agentic component cost and calibration. The agentic controller in Chapter 5 reuses a single locally deployed VLM for routing, sufficiency assessment, query rewriting, decomposition, answer generation, and hallucination verification. This simplifies deployment and can amortise prefill cost across calls, but it also introduces cost and calibration limits. Compared with non-agentic PPR retrieval, the agentic pipeline requires additional VLM calls, while its gains over the strongest non-agentic PPR setting are modest. The verifier and generator also share the same underlying model, so the verifier can detect answers that contradict the retrieved evidence but may miss hallucinations caused by shared model bias. Finally, we evaluate the agentic pipeline only through end-task performance and do not separately measure the accuracy or calibration of each controller decision. A more efficient controller, or a smaller model distilled for routing and grading, may improve the cost–performance trade-off.

Direct comparison to closely related graph RAG systems. We do not report numbers from HippoRAG 2, GraphRAG, or LightRAG re-runs on our datasets. The PPR formulation and the synonym-graph design are inspired by HippoRAG 2 and adapted to our multimodal chunk-augmented graph setting. A direct head-to-head comparison on identical corpora would more cleanly attribute the contribution of our adaptations, including multimodal nodes, chunk-augmented graph construction, and the agentic controller. This comparison is deferred to future work.

7.3 Future Work

Learned retrieval policy. The agentic controller of Section 5.3 adopts a deterministic finite-state design: profile escalation, query rewriting, decomposition, and verification are governed by hard iteration limits rather than by a learned policy. A natural extension is to replace this controller with a reinforcement-learning policy that learns when to terminate traversal, when to escalate, and which relational paths to prioritise. GraphRAG-R1 (Yu et al., 2026) and Graph R1 (Luo et al., 2025a) demonstrate that process-constrained outcome-based RL over knowledge graphs can discourage both shallow retrieval and excessive reasoning steps, yielding adaptive retrieval depths that the present controller's fixed budget cannot express.

Citation-aware retrieval for survey generation. The SurGE results show that abstract-level graph retrieval is not sufficient for high-coverage survey generation. Future work should integrate citation-aware signals into the graph, including citation links, co-citation structure, bibliographic metadata, venue and year information, and paper-level influence features. Full-text indexing would also help recover methodological details and related-work connections that are absent from abstracts. This would move the retrieval target from local semantic co-occurrence toward the scholarly relevance used by survey authors when selecting citations.

Learned graph reasoning. A complementary direction to the PPR traversal of Section 5.2 is to replace heuristic graph algorithms with a learned graph-reasoning model. We explored this direction during development by integrating G-Reasoner (Luo et al., 2025b),

a graph foundation model that trains a query-dependent graph neural network on diverse knowledge-graph reasoning tasks and produces a ranked list of document nodes in a single forward pass. Such a model could learn query-conditioned edge weights and implicit relational chains that PPR treats only through graph topology.

Learned entity recognition and alignment. The entity-resolution step of Chapter 4 is deliberately lightweight: type-aware identifiers and embedding-based synonym linking are training-free heuristics that operate on a single workspace without supervision. Three extensions would strengthen graph coherence. First, a dedicated NER model (Li et al., 2022) could replace prompt-based extraction for entity boundaries and types, reducing the type-assignment errors that propagate into the identifier scheme. Second, when a curated domain knowledge base is available, an entity-linking step in the style of BLINK (Wu et al., 2020) or GENRE (De Cao et al., 2021) could ground extracted entities to canonical identifiers, replacing surface-form synonym edges with verified equivalences. Third, an embedding-based entity-alignment model (Chen et al., 2017; Sun et al., 2020) could be adapted to learn cross-chunk and cross-document equivalences from the graph structure itself, rather than from cosine similarity alone, which would address the residual fragmentation that synonym edges only partially resolve (Table 6.12). The common trade-off is that each option adds a trained component or an external dependency, which must be weighed against the deterministic, privacy-preserving design adopted here.

Lightweight and calibrated controller modules. A further extension is to replace the prompted 30B-class VLM calls in the router, sufficiency grader, and hallucination verifier with dedicated small models in the 1.5B–7B parameter range. Each of these components is a short-input classification task and does not require the full generative capacity of the backbone VLM. This substitution would improve latency, reduce memory pressure, and improve calibration. Training data can be bootstrapped by logging router, grader, and verifier decisions from the current prompted pipeline and retaining traces whose final answer is verified against ground truth, yielding a self-labelled supervised dataset for fine-tuning.

Error analysis and component attribution. This thesis evaluates the system mainly through end-task metrics and does not separate the sources of retrieval failure. A systematic error analysis is a natural next step: classifying failures by stage, missing nodes or edges from graph construction, evidence present in the graph but not retrieved by PPR or ranking, and evidence retrieved but answered incorrectly at generation, would localize where accuracy is lost and guide targeted improvements. The same analysis would quantify caption-omission errors in the multimodal setting, where a visual detail absent from the VLM-generated description cannot surface at retrieval, and would measure false-insufficiency and false-groundedness rates in the agentic controller. Relatedly, the entity-resolution contribution is currently assessed through graph-structural diagnostics (Table 6.12); isolating the downstream effect of type-aware disambiguation and synonym linking through controlled on/off ablations, including a word-level disambiguation rate and the per-component QA impact, would attribute their individual contributions more precisely. These analyses were constrained by limited access to computational resources and are left to future work.

Chapter 8
Conclusion

This thesis addressed three structural limitations of graph-based multimodal RAG as instantiated in the RAG-Anything and LightRAG frameworks: knowledge graph fragmentation under surface-form entity resolution, topological underutilization caused by one-hop retrieval, and the absence of adaptive control and answer verification in single-pass pipelines.

We presented a system that integrates several components on top of the RAG-Anything multimodal foundation. A type-aware entity disambiguation step reduces erroneous entity merging, while an embedding-based synonym-linking step connects surface-form variants without merging nodes. A PPR traversal layer over a chunk-augmented knowledge graph lifts retrieval beyond the one-hop neighborhood and produces chunk-level relevance scores directly. A hybrid vector retrieval path combines BGE-M3 sparse and dense signals. A deterministic agentic controller routes queries to retrieval profiles, grades the sufficiency of retrieved evidence, verifies generated answers, and escalates through bounded refinement cycles. The complete system is deployed locally on vLLM with vision-language generators and is exposed to end users through a single-page web application in Section 3.5.

Empirically, the strongest configuration reaches 71.62% accuracy on the DocBench Academic subset, compared with 67.99% for the same generator before the final retrieval and indexing configuration. The 3.63-point gap is the portion most directly associated with the retrieval and indexing changes introduced in the final configuration. In the shared-graph DocBench setting, hybrid retrieval gives the strongest overall result for both KG and PPR retrieval paths, showing that retrieval design has a clear effect when cross-document distractors are present. On the multi-hop QA suite, PPR-based retrieval improves answer quality over the LightRAG-style KG baseline. Under the CoT-style setting, hybrid retrieval is the most consistent single component, and the combination of hybrid retrieval and synonym edges gives the strongest non-agentic PPR configuration. Among all reported multi-hop QA settings, the agentic retrieval setting with chunk $k = 10$ gives the strongest EM and F1 on all three datasets. On the SurGE subset, PPR with chunk reranking improves high-depth recall, but absolute recall remains limited; we therefore report SurGE as a boundary finding that maps the limits of the proposed approach in citation-oriented corpus-scale retrieval.

The contributions of this work are best described as compositional. Several components, in particular PPR over an entity–chunk graph and the synonym-graph idea, are adapted from HippoRAG 2 (Gutiérrez et al., 2025); the dual-level keyword retrieval is inherited from LightRAG (Guo et al., 2025b); and the multimodal ingestion is provided by RAG-Anything (Guo et al., 2025a). The added value lies in the joint design: typed entity disambiguation propagated through the entire pipeline, a chunk-augmented graph that produces chunk-level PPR scores without an entity-to-chunk projection, a multi-path retriever with profile-specific RRF weights, a deterministic closed-loop controller with an explicit upper bound on language-model invocations, and a privacy-preserving local deployment that exposes the full pipeline through a browser interface.

The broader message we draw from this work is that, in the current state of graph-augmented multimodal RAG, system-level integration is itself a research subject. Individual components can yield gains in isolation, but the gain that matters in practice is the gain delivered to a user querying through the deployed interface. That gain depends on how routing, retrieval, generation, and verification interact under realistic query distributions. The agentic controller is the part of the system most directly aimed at this interaction, and we view its further development—through learned routing, learned verification, citation-aware retrieval, and graph-neural reasoning over the same indexed structure—as the natural continuation of this line of work.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631. ACM.

Arize AI (2024). Phoenix: Ai observability and evaluation. GitHub repository. Accessed: 2026-05-26.

Asai, A., Wu, Z., Wang, Y., Sil, A., and Hajishirzi, H. (2024). Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations (ICLR)*.

BAAI (2024a). Baai/bge-m3. Hugging Face model card. Accessed: 2026-01-20.

BAAI (2024b). Baai/bge-reranker-v2-m3. Hugging Face model card. Accessed: 2026-01-20.

Bai, S. et al. (2025). Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*.

Cai, Y. and Zu, H. (2026). SurGE_subset: A controlled subset for retrieval evaluation on SurGE. Hugging Face dataset. Accessed: 2026-05-26.

CFDA and CLIP Labs (2025). ESG report processing pipeline. `https://github.com/cnclabs/website.kg.esg.demo`. Software demonstrator, CFDA & CLIP Labs (PIs: C.-J. Wang and M.-F. Tsai). Accessed: 2026-06-06.

Chandra, J., Navneet, S. K., and Zhang, Y. (2025). The hybrid multimodal graph index (HMGI): A comprehensive framework for integrated relational and vector search. *arXiv preprint arXiv:2510.10123*.

Chen, J., Xiao, S., Zhang, P., Luo, K., Lian, D., and Liu, Z. (2024). M3-embedding: Multilinguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*.

Chen, M., Tian, Y., Yang, M., and Zaniolo, C. (2017). Multilingual knowledge graph embeddings for cross-lingual knowledge alignment. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1511–1517.

Cormack, G. V., Clarke, C. L. A., and Buettcher, S. (2009). Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 758–759.

De Cao, N., Izacard, G., Riedel, S., and Petroni, F. (2021). Autoregressive entity retrieval. In *International Conference on Learning Representations (ICLR)*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*.

Edge, D., Trinh, H., Cheng, N., Bradley, J., Chao, A., Mody, A., Truitt, S., Metropolitansky, D., Ness, R. O., and Larson, J. (2025). From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.

Faysse, M., Hughes, G., Viaud, L., et al. (2024). Colpali: Efficient document retrieval with vision language models. *arXiv preprint arXiv:2407.01449*.

FlagOpen (2024). Flagembedding. GitHub repository. Accessed: 2026-01-20.

GraphRAG-Bench (2025). Graphrag-benchmark. GitHub repository. Accessed: 2026-03-02.

Guo, Z., Ren, X., Xu, L., Zhang, J., and Huang, C. (2025a). RAG-Anything: All-in-one RAG framework. *arXiv preprint arXiv:2510.12323*.

Guo, Z., Xia, L., Yu, Y., Ao, T., and Huang, C. (2025b). LightRAG: Simple and fast retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 10746–10761, Suzhou, China. Association for Computational Linguistics.

Gutiérrez, B. J., Shu, Y., Qi, W., Zhou, S., and Su, Y. (2025). From RAG to memory: Non-parametric continual learning for large language models. *arXiv preprint arXiv:2502.14802*.

He, X., Tian, Y., Sun, Y., Chawla, N. V., Laurent, T., LeCun, Y., Bresson, X., and Hooi, B. (2024). G-Retriever: Retrieval-augmented generation for textual graph understanding and question answering. In *Advances in Neural Information Processing Systems*, volume 37.

Ho, X., Duong Nguyen, A.-K., Sugawara, S., and Aizawa, A. (2020). Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics (COLING)*, pages 6609–6625.

Jeong, S., Baek, J., Cho, S., Hwang, S. J., and Park, J. (2024). Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7036–7050, Mexico City, Mexico. Association for Computational Linguistics.

Jiang, Z., Xu, F., Gao, L., Sun, Z., Liu, Q., Dwivedi-Yu, J., Yang, Y., Callan, J., and Neubig, G. (2023). Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992, Singapore. Association for Computational Linguistics.

Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. (2020). Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. (2023). Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, pages 611–626. ACM.

LangChain Team (2024). LangGraph: Building stateful, multi-actor applications with LLMs. GitHub repository. Accessed: 2026-05-26.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Li, J., Sun, A., Han, J., and Li, C. (2022). A survey on deep learning for named entity recognition. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):50–70.

Liang, L., Sun, M., Gui, Z., Zhu, Z., Jiang, Z., Zhong, L., Qu, Y., Zhao, P., Bo, Z., Yang, J., Xiong, H., Yuan, L., Xu, J., Wang, Z., Zhang, Z., Zhang, W., Chen, H., Chen, W., and Zhou, J. (2024). KAG: Boosting LLMs in professional domains via knowledge augmented generation. *arXiv preprint arXiv:2409.13731*.

Luan, Y., Eisenstein, J., Toutanova, K., and Collins, M. (2021). Sparse, dense, and attentional representations for text retrieval. *Transactions of the Association for Computational Linguistics*, 9:329–345.

Luo, H., E, H., Chen, G., Lin, Q., Guo, Y., Xu, F., Kuang, Z., Song, M., Wu, X., Zhu, Y., and Tuan, L. A. (2025a). Graph-r1: Towards agentic graphrag framework via end-to-end reinforcement learning.

Luo, L., Zhao, Z., Liu, J., Qiu, Z., Dong, J., Panev, S., Gong, C., Vu, T.-T., Haffari, G., Phung, D., Liew, A. W.-C., and Pan, S. (2025b). G-reasoner: Foundation models for unified reasoning over graph-structured knowledge. *arXiv preprint arXiv:2509.24276*.

Ma, S., Xu, C., Jiang, X., Li, M., Qu, H., Yang, C., Mao, J., and Guo, J. (2025). Think-on-graph 2.0: Deep and faithful large language model reasoning with knowledge-guided retrieval augmented generation. In *The Thirteenth International Conference on Learning Representations*.

Malkov, Y. A. and Yashunin, D. A. (2020). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836.

Mavromatis, C. and Karypis, G. (2025). GNN-RAG: Graph neural retrieval for efficient large language model reasoning on knowledge graphs. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 16682–16699.

Neo4j (2024). The GraphRAG manifesto: Adding knowledge to GenAI. `https://neo4j.com/blog/genai/graphrag-manifesto/`. Accessed: 2026-06-04.

Nogueira, R. and Cho, K. (2019). Passage re-ranking with BERT. *arXiv preprint arXiv:1901.04085*.

OpenGVLab (2024a). Opengvlab/internvl2-26b. Hugging Face model card. Accessed: 2026-02-25.

OpenGVLab (2024b). Opengvlab/internvl2-26b-awq. Hugging Face model card. Accessed: 2026-02-25.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The PageRank citation ranking: Bringing order to the web. Accessed: 2026-04-25.

Qdrant Team (2024). Qdrant: Vector similarity search engine. GitHub repository. Accessed: 2026-05-26.

Qwen (2024a). Qwen/qwen2-vl-7b-instruct. Hugging Face model card. Accessed: 2026-01-25.

Qwen (2024b). Qwen/qwen2.5-32b-instruct. Hugging Face model card. Accessed: 2026-01-26.

Qwen (2025). Qwen/qwen3-vl-30b-a3b-instruct-fp8. Hugging Face model card. Accessed: 2026-03-18.

Robertson, S. and Zaragoza, H. (2009). The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389.

Robinson, I., Webber, J., and Eifrem, E. (2015). *Graph Databases: New Opportunities for Connected Data*. O'Reilly Media, Sebastopol, CA, 2nd edition.

Sarthi, P., Abdullah, S., Tuli, A., Khanna, S., Goldie, A., and Manning, C. D. (2024). RAPTOR: Recursive abstractive processing for tree-organized retrieval. *arXiv preprint arXiv:2401.18059*.

Sevgili, Ö., Shelmanov, A., Arkhipov, M., Panchenko, A., and Biemann, C. (2022). Neural entity linking: A survey of models based on deep learning. *Semantic Web*, 13(3):527–570.

Singh, A., Ehtesham, A., Kumar, S., and Khoei, T. T. (2025). Agentic retrieval-augmented generation: A survey on agentic RAG. *arXiv preprint arXiv:2501.09136*.

Su, W., Xie, A., Ai, Q., Long, J., Chen, X., Mao, J., Ye, Z., and Liu, Y. (2026). SurGE: A benchmark and evaluation framework for scientific survey generation. *arXiv preprint arXiv:2508.15658*.

Sun, Z., Zhang, Q., Hu, W., Wang, C., Chen, M., Akrami, F., and Li, C. (2020). A benchmarking study of embedding-based entity alignment for knowledge graphs. *Proceedings of the VLDB Endowment*, 13(11):2326–2340.

Trivedi, H., Balasubramanian, N., Khot, T., and Sabharwal, A. (2022). MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Wang, B., Xu, C., Zhao, X., Ouyang, L., Wu, F., Zhao, Z., Xu, R., Liu, K., Qu, Y., Shang, F., Zhang, B., Wei, L., Sui, Z., Li, W., Shi, B., Qiao, Y., Lin, D., and He, C. (2024a). MinerU: An open-source solution for precise document content extraction. *arXiv preprint arXiv:2409.18839*.

Wang, J. and Han, J. (2025). PropRAG: Guiding retrieval with beam search over proposition paths. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 6212–6227.

Wang, P., Bai, S., Tan, S., Wang, S., Fan, Z., Bai, J., Chen, K., Liu, X., Wang, J., Ge, W., Fan, Y., Dang, K., Du, M., Ren, X., Men, R., Liu, D., Zhou, C., Zhou, J., and Lin, J. (2024b). Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Wang, S., Fang, Y., Zhou, Y., Liu, X., and Ma, Y. (2025). ArchRAG: Attributed community-based hierarchical retrieval-augmented generation. *arXiv preprint arXiv:2502.09891*.

Wang, Y., Lipka, N., Rossi, R. A., Siu, A., Zhang, R., and Derr, T. (2024c). Knowledge graph prompting for multi-document question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 19206–19214.

Wu, L., Petroni, F., Josifoski, M., Riedel, S., and Zettlemoyer, L. (2020). Scalable zero-shot entity linking with dense entity retrieval. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6397–6407.

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al. (2024). Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*. Accessed: 2026-02-16.

Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. (2018). HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2369–2380.

Yu, C., Zhao, K., Li, Y., Chang, H., Feng, M., Jiang, X., Sun, Y., Li, J., Zhang, Y., Li, J., et al. (2026). Graphrag-r1: Graph retrieval-augmented generation with process-constrained reinforcement learning. In *Proceedings of the ACM Web Conference 2026 (WWW '26)*.

Zhang, N., Choubey, P. K., Fabbri, A., Bernadett-Shapiro, G., Zhang, R., Mitra, P., Xiong, C., and Wu, C.-S. (2025a). SiReRAG: Indexing similar and related information for multihop reasoning. In *The Thirteenth International Conference on Learning Representations*.

Zhang, Q., Chen, S., Bei, Y., Yuan, Z., Zhou, H., Hong, Z., Chen, H., Xiao, Y., Zhou, C., Dong, J., Chang, Y., and Huang, X. (2025b). A survey of graph retrieval-augmented generation for customized large language models.

Zou, A., Yu, W., Zhang, H., Ma, K., Cai, D., Zhang, Z., Zhao, H., and Yu, D. (2024). Docbench: A benchmark for evaluating llm-based document reading systems.

Appendices

Appendix A
Statement of Gen-AI Use

Generative AI tools were used during this master's thesis as supporting tools. The main use cases were:

1. Generating and improving LaTeX table structures, figure captions, and formatting.
2. Finding and exploring sources, with all sources checked by the authors before use.
3. Understanding technical concepts related to the thesis by prompting AI tools for explanations.
4. Improving grammar, clarity, and readability of selected parts of the thesis text.
5. Assisting with code implementation, debugging, refactoring, and code review.

The use of generative AI tools was treated as assistance rather than as an independent source of scientific evidence. All research questions, system design choices, implementation decisions, experimental setup, result analysis, and conclusions were made, checked, and approved by the authors.

Appendix B

Prompt Templates

This appendix lists the main prompt templates used in the system. The prompts are grouped by their role in the pipeline: indexing, retrieval control, answer generation, and evaluation. Placeholders enclosed in braces, such as `{query}` and `{context_data}`, are filled by the system at runtime.

B.1 Indexing Prompts

B.1.1 Entity Type Schema

entity_type_schema

```
Entity types:
- person
- organization
- location
- event
- artifact
- work
- naturalentity
- concept
- process

Multimodal node types:
- image
- table
- equation
- generic(e.g., page number, footer, header, footnote)
```

B.1.2 Knowledge Graph Extraction Prompt

kg_extraction_prompt

```
---Role---
You are a Knowledge Graph Specialist. Your task is to extract entities and
relationships from the input text and output them in a strict format.

---Type Definitions---
Use ONLY the entity types listed in <Entity_types> in the user message.
Source-grounded means explicitly named in the source as a concrete referent-not a
local role, filler, or modifier phrase.

- person        : named individual human (real, historical, or fictional).
- organization  : named group of humans acting as a collective unit.
- location      : named geographic or spatial place.
- event         : named occurrence anchored in time, even if temporally fuzzy.
                  One-time anchored occurrence = event; repeatable workflow = process.
- artifact      : human-made physical object you can hold or point at (chip, device, specimen).
                  Touchable physical object = artifact; citable/deployable intellectual output =
     work.
- work          : named intellectual output: papers, software, datasets, models, standards,
                  regulations, specifications, reports, plans, runbooks, postmortems.
                  Text refers to a document/plan/runbook/postmortem = work; executed method/
    workflow = process.
- naturalentity : named entity existing in the physical world independently of human production
                  (gene, species, chemical element, cell line).
                  Concrete biological/physical object = naturalentity; measurable property or
    abstraction = concept.
- concept       : stable named domain concept best answered by "what IS it"
                  (Attention Mechanism, Ionic Conductivity, Myocardial Fibrosis).
                  WHAT = concept; HOW = process; when both apply, prefer process.
- process       : named/reusable method, workflow, procedure, or analysis activity best
                  answered by "how is it done" (Gradient Descent, CRISPR-Cas9, Homologous
    Recombination).

Use ONLY the types provided in <Entity_types>. No other type values are permitted.

---What Must Never Be Extracted as an Entity---
Never extract the following as entity nodes:
- metric values (92.3%, 14ms, $2.4M): embed in descriptions instead.
- role titles (CEO, Director, Engineer): embed in person description and relation description.
- unnamed generics (a model, the query, retrieved documents, inputs, outputs, results): skip
    entirely.
- pure time labels (Q3 2024, FY2023, deadline): skip unless part of a full named entity
  (e.g. "Q3 2024 Product Review" = event, "Q3 2024 Root Cause Analysis" = process).
- file/layout noise (paths, filenames, page numbers, bbox, chunk IDs): skip entirely.
- negated entities: extract them normally as entities.
- negated relations: do NOT create a positive relation edge; note in entity description instead.

---Negation Rule---
  "GPT-3 was trained WITHOUT RLHF"
  -> Extract both GPT-3 (work) and RLHF (process) as entities.
  -> Do NOT create a GPT-3 -[trained_with]-> RLHF relationship edge.
  -> In the GPT-3 description, note: "[negated context: does not use RLHF here]"

---Canonical Type Rule---
One surface name -> one entity type, stable across the entire extraction run.
Never output the same surface name with two different types.
If an entity has dual identity, assign the type matching its PRIMARY FUNCTION
in the world and hold it globally.

---Ambiguity Protocol---
Use only when a candidate's validity or type is unclear.
  1) If the phrase is only a descriptor, modifier, role, or generic placeholder
     ("the query", "retrieved documents", "the generator"), DO NOT EXTRACT.
  2) For stable named referents, use the WHAT/HOW test:
     "X is ..." -> concept; "X works by ..." -> process; if both apply, prefer
```

```
     process; if neither applies, DO NOT EXTRACT.
  3) Keep only candidates that are central to the passage and can participate in
     source-supported relationships.

Generic roles, unnamed placeholders, and modifier phrases belong in
relationship_description, not as entity nodes.
A correctly dropped entity is always preferable to a wrongly typed one.

---Depth-1 Rule---
Extract only the outermost complete named entity.
  "EU AI Act Article 13" -> extract "EU AI Act" (work), not "Article 13" alone.
Exception: extract a sub-component only when it is independently and widely
referenced by that name outside this document.

---Entity Output Format---
Output one line per entity. Fields are separated by {tuple_delimiter}.
The first field must be the literal string ‘entity‘.

  entity{tuple_delimiter}entity_name{tuple_delimiter}entity_type{tuple_delimiter}
     entity_description

  entity_name        : Preserve source lexical content from the text.
                       {entity_name_case_rule}
                       Do NOT rewrite aliases or abbreviations:
                       "RAG" must not become "Retrieval-Augmented Generation"
                       unless that full lexical form appears in the source text.
                       Consistent naming across the entire extraction run.
  entity_type        : Lowercase. Must be one of the types in <Entity_types>.
                       No other values permitted.
  entity_description : 1-2 objective sentences in third person.
                       Based solely on information present in the input text.
                       Do not introduce knowledge not found in the source text.
                       If context is limited, use neutral wording such as
                       "Entity mentioned in text with limited context."
                       For person entities: MUST include any title or role mentioned
                       in the input text. Format: "[name], [role] at [org] per the text."

---Relationship Output Format---
Output one line per relationship. Fields are separated by {tuple_delimiter}.
The first field must be the literal string ‘relation‘.

  relation{tuple_delimiter}source_entity{tuple_delimiter}target_entity{tuple_delimiter}
     relationship_keywords{tuple_delimiter}relationship_description

  source_entity / target_entity : Must match entity_name exactly as extracted above.
                                  {relation_endpoint_case_rule}
  relationship_keywords         : One or more high-level keywords. Separate with comma.
                                  Do NOT use {tuple_delimiter} inside this field.
                                  Use lowercase by default; preserve meaningful
                                  mixed/uppercase proper nouns and acronyms.
  relationship_description      : Concise explanation based solely on the input text.
                                  Do not add external background knowledge.
                                  Embed metric values and role titles here as natural
                                  language rather than extracting them as entities.

---Extraction Workflow---
Follow this process before writing the final output.

1. Identify central, source-supported entities first.
   Prefer entities that participate in the passage’s main claims, methods,
   components, datasets, evaluations, causes, results, or comparisons.
   Do not extract a weak entity only because it appears once.

2. Build a candidate relation set from explicit statements and direct,
   source-supported implications in the input.
   Metric values and role titles remain attributes in descriptions, not entity endpoints.
   Negated facts do not produce positive relation edges.

3. Convert multi-part claims into binary relations.
```

```
4. Enforce endpoint closure.
   Every relation endpoint must be present as an entity in the current extraction result.
   If a candidate relation has a valid source-supported endpoint that is absent, add that entity
     .
   If the endpoint is not valid under the entity rules, delete the candidate relation.

5. Prune unsupported edges.
   Do not add a relation only to reduce graph islands.
   Sparse output is correct when the source does not state enough relational evidence.

---General Instructions---
1. Output all entities first, then all relationships.
2. Relationships are undirected unless stated otherwise.
3. Decompose N-ary relationships into binary pairs.
4. Write all entity names and descriptions in the third person.
5. Output language: {language}.
6. After all entities and relationships are output, write the completion signal:
   {completion_delimiter}

---Examples---
{examples}
```

B.1.3 Multimodal Analysis Prompt

multimodal_analysis_prompt

```
CRITICAL INSTRUCTION: Your response MUST be a valid JSON object only. Start with {{ and end with
      }}. No text before or after.

Please analyze this image in detail, considering the surrounding context. Provide a JSON
     response with the following structure:

{{
    "detailed_description": "A comprehensive and detailed visual description of the image
     following these guidelines:
    - Describe the overall composition and layout
    - Identify all objects, people, text, and visual elements
    - Explain relationships between elements and how they relate to the surrounding context
    - Note colors, lighting, and visual style
    - Describe any actions or activities shown
    - Include technical details if relevant (charts, diagrams, etc.)
    - Reference connections to the surrounding content when relevant
    - Always use specific names instead of pronouns",
    "entity_info": {{
        "entity_name": "{entity_name}",
        "entity_type": "image",
        "summary": "concise summary of the image content, its significance, and relationship to
     surrounding content (max 100 words)"
    }}
}}

Context from surrounding content:
{context}

Image details:
- Image Path: {image_path}
- Captions: {captions}
- Footnotes: {footnotes}

Focus on providing accurate, detailed visual analysis that incorporates the context and would be
      useful for knowledge retrieval.

FINAL REMINDER: Output ONLY the JSON object. Do not add explanations, commentary, or markdown
     code blocks.
```

B.2 Retrieval Prompts

B.2.1 Keyword Extraction Prompt

keywords_extraction

```
---Role---
You are an expert keyword extractor, specializing in analyzing user queries for a Retrieval-
     Augmented Generation (RAG) system. Your purpose is to identify both high-level and low-level
      keywords in the user's query that will be used for effective document retrieval.

---Goal---
Given a user query, your task is to extract two distinct types of keywords:
1. high_level_keywords: overarching themes and relational concepts that describe the query's
     intent. Use lowercase short phrases (e.g., "gene knockout", "performance metrics", "api
     integration").
2. low_level_keywords: specific named entities mentioned or implied by the query. Use title-
     cased style. Covers any named referent of these types: person, organization, location, event
     , artifact, work, naturalentity, concept, process.

---Instructions & Constraints---
1. Output Format: Your output MUST be a valid JSON object and nothing else. Do not include any
     explanatory text, markdown code fences, or any other text before or after the JSON. It will
     be parsed directly by a JSON parser.
2. Source of Truth: Keywords must be grounded in the user query. Direct mentions are always
     included; reasonable inferences are permitted.
3. Concise & Meaningful: Keywords should be concise words or meaningful phrases. Prioritize
     multi-word phrases when they represent a single concept.
4. Handle Edge Cases: For queries that are too simple, vague, or nonsensical, return empty lists
      for both keyword types.
5. Language: All extracted keywords MUST be in {language}. Proper nouns should be kept in their
     original language.
6. Casing (high_level_keywords): Use lowercase phrases by default. Preserve meaningful uppercase
      or mixed-case proper nouns/acronyms.
7. Casing (low_level_keywords): Use entity-style casing. Preserve mixed-case proper nouns/
     acronyms; otherwise normalize case-insensitive phrases to canonical title-style wording.
8. Exclusions: Do not include raw metric values, standalone role titles without a name, unnamed
     generic placeholders, or standalone time labels unless part of a full named entity.
9. Complete Names: For low_level_keywords, use the outermost complete named entity.

---Examples---
{examples}

---Real Data---
User Query: {query}

---Output---
Output:
```

B.2.2 PPR Recognition Prompt

PPR_recognition_prompt

```
You are an entity relevance judge.

Query: {query}

Below are retrieved entities (with descriptions) and relational facts. Select ONLY those entity
     identifiers directly relevant to answering the query.
You MUST copy each entity identifier EXACTLY as it appears (including any "|TYPE" suffix). Do
     not rephrase, abbreviate, or invent new identifiers.
```

```
Entities:
{entity_lines_or_none}

Retrieved facts (src | relation | tgt):
{fact_lines_or_none}

Return up to {linking_top_k} relevant entity identifiers, one per line. Output ONLY the
    identifier (the part before ':' if a description follows). Entities appearing as src or tgt
    endpoints in the facts section are also valid candidates. If none are relevant, return an
    empty response.
```

B.2.3 Agentic Retrieval Prompts

classifier_prompt

```
You are a retrieval routing classifier. Select exactly one retrieval profile for
the user query. Output JSON only.

Available profiles:

- precise: Exact lexical matching over chunks.
  Use for IDs, error codes, CVEs, order numbers, version numbers, exact titles,
  code names, table names, quoted strings, or rare strings that must match text.
  A person, organization, or place name alone is NOT precise.

- semantic: Default dense+sparse chunk retrieval for ordinary RAG questions.
  Use when the answer is likely in semantically similar passages and graph
  structure is not clearly required.

- local: KG entity-neighbour retrieval.
  Use only for one focal entity and its direct attributes, direct neighbours, or
  direct relationships.

- global: KG edge/relation retrieval.
  Use for relationship, event, theme, participation, interaction, acquisition,
  affiliation, dependency, or other relation-centric questions where the edge is
  more important than one focal entity.

- multihop: PPR graph retrieval.
  Use for bridge, composition, comparison, causal chains, shared intermediate
  entities, or cross-document reasoning. Multiple names alone are not enough;
  the query must require combining facts.

Disambiguation rules:
- Do not use local/global/multihop unless graph structure is clearly useful.
- Prefer semantic for ordinary factual, definition, summary, and procedure questions, even if
    they mention a named entity.
- Prefer local over multihop for one entity and one direct property/relation.
- Prefer global over local when the query is about a relation/event type rather than a single
    entity.
- Prefer multihop when the answer requires connecting two or more facts.

Output format:
{{"reasoning": "<one short sentence>", "profile": "<name>", "confidence": <0.0-1.0>}}
{avoid_instruction}
Query: {query}
```

grader_prompt

```
Question: {query}

Task:
```

```
1. Are the chunks above sufficient to accurately answer the question?
2. If not sufficient, is this because the information genuinely does not exist in these
     documents
(unanswerable), rather than because retrieval was incomplete?

Output one compact JSON object only:
{{"sufficient": true|false, "unanswerable": true|false, "reason": "<one short sentence>"}}

Rules for "unanswerable":
- Set true ONLY when the available chunks explicitly state or clearly imply the requested
  information is not present in the indexed documents.
- Set false when chunks are empty or off-topic; that may simply mean retrieval needs improvement
     .
- When in doubt, set false (prefer retrying over giving up early).

Rules for "sufficient":
- Set true only when the chunks contain all facts needed to answer the exact question.
- For multi-hop, comparison, or causal questions, every bridge fact must be present.
- If an entity is present but the required relation or supporting passage is missing, set false.
```

rewriter_prompt

```
The following query did not retrieve sufficient evidence.

Original query: {query}
Retrieval feedback: {reason}

Rewrite the query to improve retrieval. Strategies:
- Replace ambiguous terms with synonyms
- Add explicit domain context
- Decompose compound noun phrases

Output the rewritten query only. No explanation, no quotation marks.
```

decompose_prompt

```
Break this question into {max_sub} or fewer independent sub-questions, each answerable by
     searching a knowledge base independently.

Question: {query}

Rules:
- Each sub-question must be self-contained (no references to other sub-questions).
- Sub-questions must not overlap in scope.
- Prefer 2 sub-questions over 4 unless truly needed.

Output JSON: {{"sub_questions": ["...", "..."]}}
```

hallucination_checker_prompt

```
Answer: {answer}

Question being answered: {query}

For every factual claim in the Answer, verify it is explicitly supported by the Context above.
Only mark grounded=true if each factual claim is directly supported by the supplied chunks.
Statements such as "I cannot determine X from the context" make no factual claims and are
     grounded.

Output JSON only:
{{
  "grounded": true|false,
  "ungrounded_claims": ["<claim>", ...]
```

```
}}
```

B.3 Answer Generation Prompts

B.3.1 KG RAG Prompt

kg_rag_response

```
---Role---

You are an expert AI assistant specializing in synthesizing information from a provided
     knowledge base. Your primary function is to answer user queries accurately by ONLY using the
      information within the provided Context.

---Goal---

Generate a comprehensive, well-structured answer to the user query.
The answer must integrate relevant facts from the Knowledge Graph and Document Chunks found in
     the Context.
Consider the conversation history if provided to maintain conversational flow and avoid
     repeating information.

---Instructions---

1. Step-by-Step Instruction:
  - Carefully determine the user's query intent in the context of the conversation history.
  - Scrutinize both `Knowledge Graph Data` and `Document Chunks` in the Context.
  - Weave the extracted facts into a coherent and logical response. Your own knowledge must ONLY
      be used to formulate fluent sentences and connect ideas, NOT to introduce any external
     information.
  - Track the reference_id of the document chunks which directly support the facts.
  - Generate a references section at the end of the response.
  - The References section is MANDATORY.
  - Do not generate anything after the reference section.

2. Content & Grounding:
  - Strictly adhere to the provided context.
  - If the answer cannot be found in the Context, state that you do not have enough information
     to answer.

3. Formatting & Language:
  - The response MUST be in the same language as the user query.
  - The response MUST utilize Markdown formatting.
  - The response should be presented in {response_type}.

4. References Section Format:
  - The References section should be under heading: `### References`
  - Reference list entries should adhere to the format: `* [n] Document Title`.
  - The Document Title must be taken VERBATIM from the `Reference Document List`.
  - Provide maximum of 5 most relevant citations.
  - Do not generate footnotes section or any comment after the references.

5. Reference Section Example:
### References

- [1] Document Title One
- [2] Document Title Two
- [3] Document Title Three

6. Additional Instructions: {user_prompt}

---Context---
```

```
{context_data}
```

B.3.2 Naive RAG Prompt

naive_rag_response

```
---Role---

You are an expert AI assistant specializing in synthesizing information from a provided
     knowledge base. Your primary function is to answer user queries accurately by ONLY using the
      information within the provided Context.

---Goal---

Generate a comprehensive, well-structured answer to the user query.
The answer must integrate relevant facts from the Document Chunks found in the Context.
Consider the conversation history if provided to maintain conversational flow and avoid
     repeating information.

---Instructions---

1. Step-by-Step Instruction:
  - Carefully determine the user's query intent in the context of the conversation history.
  - Scrutinize 'Document Chunks' in the Context.
  - Weave the extracted facts into a coherent and logical response. Your own knowledge must ONLY
      be used to formulate fluent sentences and connect ideas, NOT to introduce any external
     information.
  - Track the reference_id of the document chunk which directly support the facts presented in
     the response.
  - Generate a References section at the end of the response.
  - Each reference document must directly support the facts presented in the response.
  - Do not generate anything after the reference section.

2. Content & Grounding:
  - Strictly adhere to the provided context.
  - If the answer cannot be found in the Context, state that you do not have enough information
     to answer.

3. Formatting & Language:
  - The response MUST be in the same language as the user query.
  - The response MUST utilize Markdown formatting.
  - The response should be presented in {response_type}.

4. References Section Format:
  - The References section should be under heading: '### References'
  - Reference list entries should adhere to the format: '* [n] Document Title'.
  - The Document Title must be taken VERBATIM from the 'Reference Document List'.
  - Provide maximum of 5 most relevant citations.
  - Do not generate footnotes section or any comment after the references.

5. Reference Section Example:
### References

- [1] Document Title One
- [2] Document Title Two
- [3] Document Title Three

6. Additional Instructions: {user_prompt}

---Context---

{context_data}
```

B.3.3 Multi-hop CoT Answer Prompt

cot_qa_prompt

```
As an advanced reading comprehension assistant, your task is to analyze the provided passages
     and answer the corresponding question meticulously. Use only the information in the provided
      passages. Your response should start after "Thought: ", where you briefly identify the
     evidence needed to answer the question. Conclude with "Answer: " to present a concise,
     definitive response, devoid of additional elaborations.
```

B.4 Docbench Evaluation Prompt

docbench_evaluation_prompt

```
You are an expert evaluator tasked with assessing the accuracy of answers generated by a RAG (
     Retrieval-Augmented Generation) system.

Task: Evaluate whether the generated answer correctly responds to the given question based on
     the expected answer.

Question: {{question}}

Expected Answer: {{ref_ans}}

Generated Answer: {{sys_ans}}

Evaluation Criteria:
1. Accuracy (0 or 1): Does the generated answer match the factual content of the expected answer
     ?
- 1: The generated answer is factually correct and aligns with the expected answer
- 0: The generated answer is factually incorrect or contradicts the expected answer

Instructions:
- Focus on factual correctness, not writing style or format
- Consider partial matches: if the generated answer contains the correct information but
     includes additional context, it should still be considered accurate
- For numerical answers, check if the values match or are equivalent
- For list answers, check if all key elements are present
- If the expected answer is "Not answerable" and the generated answer indicates inability to
     answer, consider it accurate

Output Format:
Please respond with a JSON object containing only:
{
  "accuracy": 0 or 1,
  "reasoning": "Brief explanation of your evaluation"
}
```

Appendix C
Retrieval Configuration

This appendix reports the retrieval budgets and context settings used in the main evaluation experiments. The names in the tables follow the argument names used by the evaluation scripts. Configuration parameters are typeset in monospace font for readability. For standard KG retrieval, `chunk_top_k` controls the number of chunks kept after retrieval and reranking. For PPR retrieval, `ppr_top_k` controls the PPR candidate pool, while `ppr_qa_top_k` controls the number of PPR-ranked chunks passed to the answer stage.

For KG retrieval, textual entity and relation evidence is constrained by `max_entity_tokens`=6000 and `max_relation_tokens`=8000. In the truncated KG setting, chunks are collected only from the retained entity and relation evidence after token truncation. In the untruncated setting, chunks are collected from the full retrieved entity and relation set before token truncation. PPR retrieval does not use this truncated/untruncated KG evidence switch.

The main synonym-edge experiments use a cosine threshold of 0.80 unless otherwise stated. The synonym-threshold sensitivity experiment in Appendix F.1 varies this threshold from 0.70 to 0.90. The PPR hyperparameter optimization experiment in Appendix F.2 uses Optuna to tune `top_k`, `ppr_qa_top_k`, `ppr_top_k`, `passage_node_weight`, `ppr_damping`, and `hub_penalty_threshold`.

Table C.1: Retrieval configuration used in the main experiments. The parameter `naive_top_k` is used only for the multi-hop QA experiments. For PPR retrieval, `ppr_qa_top_k` controls the number of PPR-ranked chunks kept for downstream use. The context budget is fixed to `max_total_tokens`=45,000 across all experiment groups.

Experiment group	`top_k`	`chunk_top_k`	`naive_top_k`	`ppr_top_k`	`ppr_qa_top_k`	`min_rerank_score`	`multimodal_top_k`
DocBench	20	10	—	50	10	0.3	3
SurGE query-level	20	50	—	100	50	0.0	—
SurGE survey-level	20	500	—	750	500	0.0	—
Multi-hop QA	10	5	10	50	5	0.3	—
Agentic multi-hop, chunk $k = 5$	10	5	10	50	5	0.3	—
Agentic multi-hop, chunk $k = 10$	10	10	10	50	10	0.3	—

Table C.2: Default PPR retrieval parameters used in the main experiments. These settings are shared across DocBench, SurGE, and multi-hop QA unless explicitly overridden in the result tables. The RRF constant is used only by variants that explicitly enable RRF fusion.

PPR parameter	Value
`ppr_damping`	0.50
`passage_node_weight`	0.05
`recognition_top_k`	20
`linking_top_k`	5
`hub_penalty_threshold`	50
`ppr_post_rerank_rrf_k`	60

Appendix D
DocBench Component Sensitivity

Tables D.1 and D.2 report removal-style component sensitivity results on DocBench Academic. These results are provided for completeness. They are not used as the main DocBench comparison because the effects are not strictly monotonic: removing a component can sometimes improve the final score.

Table D.1: DocBench Academic accuracy under KG retrieval component ablations. Best values in each column are shown in bold. "w/o" denotes removing the corresponding component from the full PPR retrieval setting.

Experiment	Overall	text	mm-t	meta	mm-f	unans	web
Full KG retrieval	**58.42**	**80.33**	71.30	**8.33**	78.57	30.00	0.00
w/o chunk-only message	54.13	73.77	70.43	2.08	78.57	13.33	0.00
w/o KG rerank	57.10	77.05	70.43	**8.33**	76.19	30.00	0.00
w/o per-keyword retrieval	**58.42**	73.77	**72.17**	**8.33**	**83.33**	**33.33**	0.00
w/o hybrid retrieval	55.45	77.05	66.09	4.17	80.95	30.00	0.00

Table D.2: DocBench Academic accuracy under PPR ablations. Best values in each column are shown in bold. "w/o" denotes removing the corresponding component from the full PPR retrieval setting.

Experiment	Overall	text	mm-t	meta	mm-f	unans	web
Full PPR retrieval	57.43	73.77	73.91	12.50	73.81	23.33	0.00
w/o per-keyword retrieval	**59.74**	**77.05**	75.65	10.42	**83.33**	23.33	0.00
w/o chunk rerank	57.43	73.77	75.65	8.33	76.19	20.00	0.00
w/o hybrid retrieval	57.10	73.77	75.65	6.25	78.57	16.67	0.00
w/o synonym edges	**59.74**	73.77	**76.52**	**14.58**	78.57	**26.67**	0.00

Appendix E

Additional Multi-hop QA Component Results

This appendix reports additional multi-hop QA experiments from an earlier development stage. These experiments reuse the retrieval components first designed for the DocBench shared-graph setting. They are included for completeness, but they are not used as the main multi-hop results in Chapter 6.

There are two differences between these exploratory runs and the final multi-hop experiments. First, these runs use the original RAG answer prompt. The later main multi-hop experiments use a CoT-style answer prompt, which is more suitable for questions that require multi-step reasoning. Second, the component settings in this appendix were transferred from the DocBench experiments. They were not tuned specifically for multi-hop QA. As a result, some component additions and removal-style ablations do not lead to clear gains.

Tables E.1 and E.2 report component-addition experiments under the original prompt. They show that some retrieval changes improve recall, but the improvements do not always transfer to EM and F1. Tables E.3 and E.4 report removal-style sensitivity results. These results are mainly used to inspect component interactions. They should not be read as the final ablation evidence for the proposed multi-hop retrieval setting. The final multi-hop analysis is reported in Section 6.4.3, where the CoT-style prompt is used.

Table E.1: Multi-hop QA results under KG retrieval component additions. Best values in each column are shown in bold.

Added component	2WikiMultiHopQA				HotpotQA				MuSiQue			
	EM	F1	R@2	R@5	EM	F1	R@2	R@5	EM	F1	R@2	R@5
None (KG baseline)	0.3990	0.4872	0.6168	0.6320	0.4600	0.5986	0.7680	0.8235	0.1530	0.2534	0.3008	0.3367
Per-keyword retrieval	0.3550	0.4328	**0.6392**	**0.6555**	0.4140	0.5447	0.7540	0.8040	0.1110	0.2094	0.3164	0.3508
KG rerank	0.4040	0.4963	0.6168	0.6320	0.4510	0.5871	0.7655	0.8205	0.1520	0.2555	0.2993	0.3347
Hybrid retrieval	**0.4170**	**0.5091**	0.6355	0.6528	**0.4890**	**0.6239**	**0.8025**	**0.8640**	**0.1700**	**0.2792**	**0.3210**	**0.3577**
Untruncated KG evidence	0.4000	0.4944	0.6168	0.6320	0.4450	0.5848	0.7655	0.8205	0.1450	0.2468	0.2988	0.3347
Chunk-only message	0.3250	0.3921	0.6168	0.6320	0.3940	0.5363	0.7680	0.8235	0.1050	0.1700	0.3008	0.3367

Table E.2: Multi-hop QA results under PPR retrieval component additions. Best values in each column are shown in bold.

Added component(s)	2WikiMultiHopQA				HotpotQA				MuSiQue			
	EM	F1	R@2	R@5	EM	F1	R@2	R@5	EM	F1	R@2	R@5
None (PPR baseline)	**0.4950**	**0.5958**	0.7312	0.9125	**0.4820**	**0.6236**	0.6775	0.8650	0.1730	0.2788	0.3826	0.5210
Synonym edges	0.4790	0.5818	0.7358	0.9135	0.4460	0.5861	0.6800	0.8610	0.1700	0.2777	0.3847	0.5258
Per-keyword retrieval	0.4730	0.5802	0.7352	0.9103	0.4420	0.5829	0.6655	0.8525	0.1640	0.2738	0.3915	0.5330
Hybrid retrieval	0.4670	0.5789	**0.7578**	**0.9313**	0.4430	0.5929	0.7135	0.8965	**0.1770**	**0.2969**	**0.4041**	**0.5483**
Chunk rerank	0.3670	0.4362	0.6508	0.6723	0.4310	0.5817	**0.8270**	**0.8970**	0.1130	0.1831	0.3304	0.3727
Chunk rerank + RRF	0.3690	0.4431	0.6468	0.6735	0.4430	0.5861	0.7945	0.8955	0.1150	0.1847	0.3372	0.3693

Table E.3: Multi-hop QA ablation results under the KG retrieval path. Best values in each column are shown in bold.

Method	2WikiMultiHopQA				HotpotQA				MuSiQue			
	EM	F1	R@2	R@5	EM	F1	R@2	R@5	EM	F1	R@2	R@5
Full KG retrieval	0.361	0.4306	**0.6450**	**0.6610**	0.426	0.5707	0.7895	0.8460	0.106	0.1699	0.3297	0.3627
w/o chunk-only message	**0.394**	**0.4762**	**0.6450**	**0.6610**	**0.447**	**0.5857**	0.7895	0.8460	**0.137**	**0.2427**	0.3297	0.3627
w/o KG rerank	0.361	0.4301	**0.6450**	**0.6610**	0.412	0.5602	0.7905	0.8460	0.102	0.1676	**0.3302**	0.3628
w/o per-keyword retrieval	0.341	0.4125	0.6352	0.6525	0.424	0.5733	**0.8020**	**0.8645**	0.108	0.1749	0.3223	0.3578
w/o hybrid retrieval	0.355	0.4246	0.6392	0.6555	0.385	0.5282	0.7540	0.8040	0.092	0.1583	0.3153	0.3508

Table E.4: Multi-hop QA ablation results under the PPR retrieval path. Best values in each column are shown in bold.

Method	2WikiMultiHopQA				HotpotQA				MuSiQue			
	EM	F1	R@2	R@5	EM	F1	R@2	R@5	EM	F1	R@2	R@5
Full PPR retrieval	0.360	0.4355	0.6508	0.6733	0.443	**0.5936**	0.8270	0.8985	0.114	0.1880	0.3377	0.3801
w/o per-keyword retrieval	0.369	0.4394	0.6508	0.6733	0.436	0.5874	**0.8280**	0.8990	0.120	0.1916	0.3362	0.3792
w/o chunk rerank	**0.464**	**0.5745**	**0.7562**	**0.9207**	**0.446**	0.5904	0.6890	0.8825	**0.172**	**0.2772**	**0.4101**	**0.5473**
w/o hybrid retrieval	0.364	0.4366	0.6512	0.6727	0.432	0.5843	0.8270	0.8960	0.114	0.1834	0.3322	0.3749
w/o synonym edges	0.359	0.4303	0.6508	0.6733	0.435	0.5877	**0.8280**	**0.9000**	0.115	0.1851	0.3381	0.3803

Appendix F
Additional Retrieval Analysis

This appendix reports additional retrieval analysis that supports the main results. The experiments include synonym-edge threshold sensitivity and PPR hyperparameter optimization. These results are not used as the main benchmark comparison, but they help explain how key retrieval settings are selected.

F.1 Synonym-Edge Threshold Sensitivity

Table F.1 reports the sensitivity of PPR retrieval with hybrid retrieval and synonym edges to the synonym-edge cosine threshold. The threshold controls which entity pairs are connected by synthetic synonym edges. A lower threshold creates denser synonym connectivity but may introduce noisy edges. A higher threshold is more conservative and keeps only stronger synonym candidates.

F.2 PPR Hyperparameter Selection

We tune the PPR retrieval hyperparameters with Optuna rather than manual grid search. The search is conducted on the multi-hop QA development setting for the PPR retrieval path with hybrid retrieval and synonym edges enabled. The optimized hyperparameters include the initial retrieval depth `top_k`, the final PPR QA chunk window `ppr_qa_top_k`, the PPR candidate pool size `ppr_top_k`, `passage_node_weight`, `ppr_damping`, and `hub_penalty_threshold`.

The Optuna study uses a Tree-structured Parzen Estimator (TPE) sampler with a fixed random seed for reproducibility (Akiba et al., 2019). An anchor configuration corresponding to the pre-tuning PPR hybrid-synonym setting is enqueued before the search. This allows the optimizer to compare sampled configurations against the original setting. Each trial is

Table F.1: Multi-hop QA results for PPR with hybrid retrieval and synonym edges under different synonym-edge thresholds. Best values in each column are shown in bold.

Synonym-edge threshold	2WikiMultiHopQA				HotpotQA				MuSiQue			
	EM	F1	R@2	R@5	EM	F1	R@2	R@5	EM	F1	R@2	R@5
0.70	0.616	0.7090	**0.7660**	0.9357	0.550	0.6957	0.7120	0.8985	0.272	0.3824	0.3990	0.5483
0.75	0.610	0.7056	0.7602	**0.9375**	0.562	0.7022	**0.7185**	**0.8995**	0.278	0.3886	0.4008	0.5516
0.80	**0.629**	**0.7229**	0.7592	0.9340	**0.564**	**0.7068**	0.7150	**0.8995**	**0.283**	**0.3994**	0.4024	**0.5533**
0.85	0.621	0.7113	0.7615	0.9327	0.553	0.6932	0.7090	0.8920	0.275	0.3816	0.3975	0.5482
0.90	0.606	0.7048	0.7588	0.9355	0.552	0.6942	0.7115	0.8975	0.280	0.3951	**0.4052**	0.5532

Table F.2: Top-ranked completed Optuna trials for PPR retrieval with hybrid retrieval and synonym edges. Trials are ranked by macro-F1 across 2WikiMultiHopQA, HotpotQA, and MuSiQue.

Rank	Trial	Macro-F1	Macro-EM	R@2	R@5	`top_k`	`ppr_qa_top_k`	`ppr_top_k`	`Passage wt.`	`Damping / Hub`
1	30	**0.6694**	**0.5533**	0.6410	0.7935	40	10	25	0.10	0.65 / 0
2	18	0.6648	0.5483	0.6261	0.7827	20	10	25	0.05	0.65 / 10
3	23	0.6628	0.5450	0.6329	0.7937	20	10	25	0.10	0.65 / 10
4	10	0.6606	0.5383	0.6222	0.7764	10	10	25	0.05	0.65 / 10
5	36	0.6594	0.5450	0.6186	0.7922	10	10	25	0.10	0.65 / 1
6	39	0.6589	0.5383	0.6283	0.7935	40	10	25	0.05	0.65 / 25
7	17	0.6570	0.5317	0.6231	0.7806	20	10	50	0.05	0.65 / 10
8	32	0.6565	0.5450	0.6410	0.7943	40	10	25	0.10	0.65 / 0
9	13	0.6564	0.5383	**0.6469**	**0.8022**	20	10	25	0.05	0.50 / 0
10	2	0.6561	0.5383	0.6269	0.7853	20	10	25	0.02	0.50 / 50

evaluated on 2WikiMultiHopQA, HotpotQA, and MuSiQue. The primary optimization objective is macro-averaged F1 across the three datasets. Macro EM, Recall@2, and Recall@5 are logged as secondary diagnostic metrics.

After the development search, the top-ranked configurations are re-evaluated on verification and full evaluation settings. Table F.2 reports the top Optuna trials from the development search. Table F.3 reports the corresponding verification and full-set reruns for the strongest candidates.

Table F.3: Verification and full-set reruns for selected high-ranking Optuna configurations. Macro-F1 is used as the primary selection criterion, while macro EM and retrieval recall are reported as secondary diagnostics.

Split	Trial	Macro-F1	Macro-EM	R@2	R@5	`top_k`	`ppr_qa_top_k`	`ppr_top_k`	`Passage wt.`	`Damping` / `Hub`
Verify	trial_0018	**0.6405**	**0.5300**	0.6083	0.7800	20	10	25	0.05	0.65 / 10
Verify	trial_0030	0.6354	0.5233	**0.6246**	**0.7918**	40	10	25	0.10	0.65 / 0
Verify	trial_0036	0.6329	0.5244	0.6080	0.7809	10	10	25	0.10	0.65 / 1
Verify	anchor	0.5998	0.5000	0.6169	0.7850	10	5	50	0.05	0.50 / 50
Full	trial_0018	**0.6354**	**0.5150**	0.6156	0.7915	20	10	25	0.05	0.65 / 10
Full	trial_0030	0.6294	0.5103	**0.6293**	**0.8029**	40	10	25	0.10	0.65 / 0
Full	trial_0036	0.6278	0.5117	0.6188	0.7986	10	10	25	0.10	0.65 / 1
Full	anchor	0.6020	0.4843	0.6238	0.7934	10	5	50	0.05	0.50 / 50

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2026-06-04

EXAMENSARBETE Graph-Augmented RAG for Multimodal Document Question Answering
STUDENTER Yaliang Cai, Haobo Zu
HANDLEDARE Pierre Nugues (LTH), Xuhao Zhang (Huawei Sweden R&D)
EXAMINATOR Xuan-Son Vu (LTH)

Ett kunskapsnätverk för bättre svar från dokument

POPULÄRVETENSKAPLIG SAMMANFATTNING **Yaliang Cai, Haobo Zu**

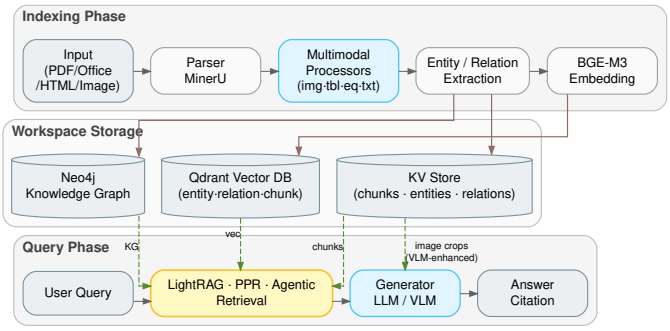
Tekniska dokument innehåller viktig information i text, tabeller och bilder. Vi har utvecklat ett lokalt AI-system som söker efter relevanta delar och kopplar samman information innan det svarar. Tester visar bättre resultat på dokumentfrågor och frågor som kräver flera informationssteg.

När man frågar om ett tekniskt dokument finns svaret inte alltid i ett enda stycke. En metod kan beskrivas i löpande text, medan det avgörande resultatet visas i en tabell, en figur eller en formel längre bak i dokumentet. Detta är vanligt i forskningsartiklar, tekniska rapporter och manualer, där olika delar av dokumentet kompletterar varandra. Ett system som bara söker efter liknande formuleringar kan därför hitta rätt ämne, men missa just det underlag som behövs för att svara korrekt.

I vårt examensarbete har vi utvecklat ett lokalt AI-system som organiserar dokumentens innehåll i en kunskapsgraf. I grafen kopplas begrepp och relationer till de textstycken, tabeller, bilder och formler där informationen finns, så att samband mellan olika delar av dokumentet blir synliga.

När en användare ställer en fråga kombineras vanlig textsökning med sökning genom grafens kopplingar. På så sätt kan systemet hitta stöd som ligger på olika platser eller nås genom en kedja av samband, snarare än bara genom ytlig ordlikhet.

Det skapar en automatiserad bro mellan textuell och visuell information, vilket ger AI-systemet en mer komplett bild av hela sammanhanget.



En extra kontrollkomponent granskar om svaret verkligen stöds av källorna, vilket minskar risken för att systemet hittar på information. All bearbetning sker lokalt, så känsliga eller affärskritiska dokument aldrig behöver skickas till externa AI-tjänster.

Vi testade systemet på akademiska dokument och på flerstegsfrågor där svaret kräver att flera fakta vägs samman. Resultaten visar att grafbaserad sökning kan ge bättre svar än traditionella metoder när informationen är spridd över ett dokument. Vid bred sökning över stora samlingar av forskningsartiklar var metoden däremot fortfarande begränsad, eftersom grafer är dyrare att bygga och söka i än enkla textindex. Arbetet visar därför både möjligheterna och gränserna för kunskapsgrafer i AI-stöd.