ENS Paris-Saclay - Amazon

Internship Thesis

Agentic Retrieval for Knowledge Bases: From Chunking to Reranking

Author:
Oliver Jack

Supervisors:
Buğra Çetingök
Samuele Compagnoni
Carlos Moyano

October 27, 2025

Abstract

This thesis studies several retrieval strategies to make enterprise knowledge bases usable for agentic systems. The analysis covers different approaches such as document chunking, graph-based multi-hop retrieval, hybrid dense-sparse search, and LLM-based reranking and how they affect performance when applied to long, technical documents. By evaluating these methods on domain-specific and open-source corpora, we provide insights into the trade-offs between accuracy, latency, and cost, offering practical guidance for building robust retrieval pipelines to support agentic applications.

I would like to personally thank Buğra Çetingök, Samuele Compagnoni and Carlos Moyano for all of their support and guidance throughout my internship.

Contents

1 Introduction

1.1 Background & Motivation

Big corporations, like Amazon, generate and depend on large corpora of technical documentation, ranging from standard operating procedures (SOPs) to research reports. Much of this content is often internal and domain-specific, meaning it was not available during pre-training of foundational models. As a result, state-of-the-art models struggle to answer business related questions that require context-specific knowledge, oftentimes leading to misinterpretations of local conventions, or even hallucinations. In parallel, many organizations, including the Supply Chain team I am part of at Amazon, are working on the implementation of their own agentic framework, with the goal of answering simple questions, performing deep dives, and taking action to automate certain tasks. For these agents to be trustworthy, it is important to rely on a well-documented knowledge base (KB), as well as a retrieval method that can return relevant information efficiently (see Figure 1 for a general overview of the agentic retrieval system).

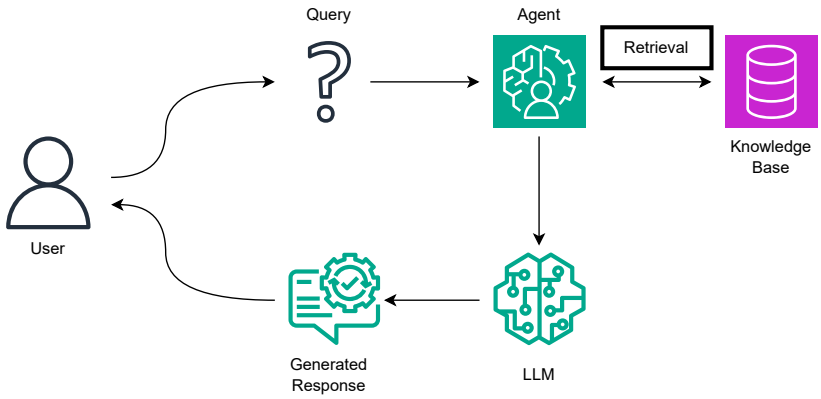

Figure 1: An agent using retrieval-augmented generation (RAG). The agent receives a user query as input, retrieves relevant chunks from the connected knowledge base, passes them to a large language model (LLM), and outputs a generated response to the user.

Retrieval-augmented generation (RAG), a concept first introduced in [Lewis et al., 2020], offers a direct way to connect large language models (LLMs) with enterprise corpora. First, the relevant passages are retrieved, before an LLM is used to generate an answer from that retrieved data. In practice, however, the quality of retrieval can fluctuate substantially, since there are several factors that make enterprise data different from publicly available data. Company documents are often times long and multimodal (including figures and tables), presenting themselves in various complex formats and layouts. They are composed of dense jargon and acronyms, while potentially requiring cross-file reasoning. Moreover, in production, dealing with such an abundance of files can have an impact on retrieval latency and high cost per query, constraining most methods that are feasible on paper. Therefore, the central question of this thesis is not whether the combination of a KB with RAG works in principle, but rather which retrieval choices, i.e. how we segment documents, exploit graph structure, combine dense and sparse signals, and rerank candidates, yield the best accuracy–efficiency trade-offs for agentic systems relying on enterprise knowledge.

1.2 Problem Statement

This thesis studies four key implementation steps that, when considered all together, determine how well a KB supports agents. The first step is how to segment long documents that have an inherent complex structure. Simple fixed-window chunking is fast and robust, but often leads to splitting paragraphs or tables into separate pieces. Semantic chunking has more awareness of headers and paragraphs, but can lead to a high variance in chunk sizes. Hierarchical chunking introduces the idea of parent-child nodes that enable top-down search. The second step is whether to introduce relational structure into retrieval. Graph-based methods such as GraphRAG [Edge et al., 2025] connect entities and documents to support multi-hop reasoning, but their benefits must be weighed against added complexity and cost. The third step focuses on how to combine sparse and dense retrieval signals. Dense encoders capture paraphrase and topic similarity [Karpukhin et al., 2020], while lexical methods such as BM25 tend to perform better for numerical retrieval [Robertson and Zaragoza, 2009]. When combining both, often referred to as hybrid fusion, retrieval accuracy may improve, yet at the expense of increased latency and cost [Thakur et al., 2021]. The fourth step is how to potentially reorder retrieved candidates. Reranking with cross-encoders [Nogueira and Cho, 2020] or LLMs [Sun et al., 2024] can improve precision at the top of the list, but may introduce unnecessary latency, potentially raising the need for a more conditional approach.

In the Supply Chain team at Amazon, we initially built a centralized knowledge base to support agentic decision-making by using standard configurations, such as fixed-window chunking with pure semantic search. This delivered functional but mediocre performance, revealing that default settings are often insufficient for enterprise requirements. This practical challenge, combined with the absence of systematic guidance on quantifying accuracy-latency-cost trade-offs for an enterprise retrieval system, motivates the need for rigorous evaluation of these four implementation steps.

1.3 Research Questions

Based on this problem statement, it would be interesting to analyze the following topics in further depth:

1. Do advanced segmentation techniques of long documents improve retrieval and answer accuracy over fixed chunking, and can agentic chunking, where LLMs guide breakpoints and enrich chunks with document-level context, provide further gains while keeping latency and cost acceptable?

2. Do knowledge graphs improve retrieval for multi-hop questions, and how does their performance compare against the best chunking baselines without graphs when tested on the same set of questions?

3. Can query-aware hybrid retrieval, where the balance between sparse and dense signals is adapted dynamically based on query features such as the length, the presence of digits or even the embeddings, achieve better accuracy than fixed-weight hybrid methods, and what is the most efficient way of learning the optimal balance between sparse and dense embeddings?

4. To what extent do rerankers based on cross-encoders or large language models improve precision when considering the top-ranked candidates, and can conditional

reranking policies, which would only apply reranking in the case of ambiguous queries, deliver significant benefit while maintaining latency low?

1.4 Thesis Structure

The remainder of this thesis is organized in the following way. Chapter 2 reviews related work on document segmentation, graph-based methods, hybrid sparse-dense retrieval, and reranking, highlighting limitations that motivate our contributions. Chapter 3 presents the actual methodology, detailing the implementation of the four research tracks mentioned above and formalizing our novel approaches: agentic chunking, fair evaluation of graph retrieval on multi-hop data, query-aware hybrid retrieval, and conditional reranking. Chapter 4 describes the experimental setup and data, including hyperparameter selection, used resources, corpus preprocessing, and questions and answers (Q&A) dataset generation. Chapter 5 reports the results for each research track, considering both retrieval accuracy, as well as latency and cost, before concluding with an "optimal" retrieval setup to improve the performance of agents. Finally, Chapter 6 summarizes the findings, discusses limitations, and outlines directions for future work.

2 Related Work

2.1 Retrieval-Augmented Generation: Components & Evolution

Modern-day RAG systems are often based on classical information retrieval (IR) foundations. Lexical methods such as BM25 [Robertson and Zaragoza, 2009] provided strong, length-normalized, exact-match scoring and remain competitive on many benchmarks. Neural retrieval became practical with Dense Passage Retrieval (DPR) [Karpukhin et al., 2020], which trained dual encoders to embed queries and passages into a shared space for efficient nearest-neighbor search. Later work improved this idea: ANCE [Xiong et al., 2020] used hard negatives from live indexes, ColBERT [Khattab and Zaharia, 2020] introduced token-level late interaction for more precise matching, Contriever [Izacard et al., 2022] showed robust unsupervised pretraining, and GTR [Ni et al., 2021] scaled encoders for better generalization. In parallel, the original RAG paper [Lewis et al., 2020] combined a neural retriever with a seq2seq generator.

Consequently, hybrid methods followed shortly after. The BEIR benchmark [Thakur et al., 2021] showed that BM25 often remains strong, especially when it comes to queries containing numbers or revolving around domain-knowledge, motivating the use of hybrid retrieval. Simple fusion techniques like Reciprocal Rank Fusion (RRF) [Cormack et al., 2009] and score-normalized combinations improved robustness at very little cost. More advanced models such as SPLADE [Formal et al., 2021] created sparse expansions to bring neural signals into inverted indexes, while HyDE [Gao et al., 2022] used an LLM to generate hypothetical documents that improved dense retrieval quality.

The next challenge consisted of adapting retrieval to long-form, structured documents. Fixed sliding windows are easy to implement but often break up coherent structure, while semantic chunking respects layout but produces uneven lengths. Hierarchical methods,

often referred to as the state-of-the-art chunking method, address this problem by building trees of chunks, enabling search at different levels of abstraction. In practice, another method would be to enrich chunks with metadata such as titles, short summaries, or entity identifiers to improve retrieval while keeping the original text as the ground truth, an idea first proposed by Anthropic [Anthropic, 2024].

Multi-hop and graph-based retrieval extend traditional RAG to queries that need evidence from multiple different sources. GraphRAG [Edge et al., 2025] builds graphs by linking entities and chunks, which can then be explored locally and globally to cover broader questions. When it comes to the ranking stage, cross-encoder models like monoBERT and MonoT5 [Nogueira and Cho, 2020, Nogueira et al., 2020] re-evaluate candidates jointly with the query, improving precision, while newer LLM-based rerankers such as RankGPT [Sun et al., 2024] have shown to offer further gains on small candidate sets.

Finally, RAG has become a key part of agentic systems. ReAct [Yao et al., 2023] was a first solution combining reasoning with retrieval, AutoGen [Wu et al., 2023] supports multi-agent collaboration with retrieval as a shared tool, and open-source frameworks like LangChain, LangGraph, and Strands simplify orchestration for tool routing and planning. Simultaneously, AWS has continued to roll out useful features, such as Amazon Bedrock Knowledge Bases and OpenSearch, supporting large-scale, enterprise indexing and retrieval.

2.2 Limitations of Current Approaches

Although the state-of-the-art methods in chunking, graph augmentation, hybrid retrieval, and reranking have developed at an impressive rate in recent years, there still exist several recurring limitations. Fixed windows are robust but have no awareness of structure, while semantic and hierarchical schemes tend to keep document layout intact but can suffer from uneven chunk sizes. Graph-based methods risk uncontrolled expansion and noisy edges, which can slow down retrieval. Hybrid retrieval generally improves accuracy but, in practice, routing that adapts to the query could be more efficient than fixed weights. Finally, rerankers improve precision but also increase latency, suggesting that they should be applied conditionally rather than blindly by default.

2.3 Main Contributions

This thesis addresses these gaps with four contributions/extensions tailored to enterprise knowledge bases and agentic RAG (see Figure 2 for a general overview of the main contributions and how they connect):

1. We introduce a new concept of chunking, which we name *agentic chunking*. The novel idea here is to allow an LLM to identify the chunk boundaries and then enrich each chunk with a contextual summary of its relevance with respect to the whole document, which is included in the embedding and can be used at retrieval time, while the ground truth text remains untouched.

2. We propose a *fair evaluation of graph-augmented retrieval*, providing a first apples-to-apples comparison of graph-based methods against the best performing non-graph baselines under identical conditions. We build graphs linking entities, chunks, and

documents, and test shallow expansions (one to two hops) under the same settings, measuring both how well all supporting data is retrieved and the trade-off with latency.

3. We develop a *query-aware hybrid retrieval mechanism*, where the idea is to find the ideal balance between keyword matching and semantic similarity, depending on the query at hand. We aim to accomplish this by learning an optimal weight per query based on features such as the number of words in the prompt or even its embedding vectors. For this, we will be testing three different implementations, each varying in complexity: ridge regression, XGBoost, and a lightweight Multilayer Perceptron (MLP).

4. We introduce the concept of *conditional reranking*, a policy that triggers LLM-reranking only when a certain level of ambiguity is detected, while reporting the precision–latency trade-off explicitly.

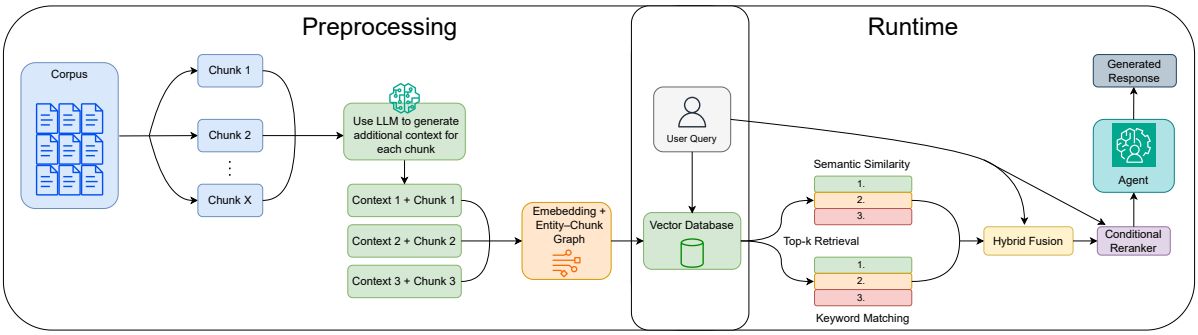

Figure 2: Summarized overview of key contributions, broken down into separate pipeline steps: chunking, graph-embedding, hybrid fusion, and reranking. The figure is split into preprocessing and runtime, highlighting which steps belong to which stage.

Beyond the individual contributions listed above, unlike most benchmark datasets that are based on sources like Wikipedia, our evaluation considers data similar to long-form enterprise corpora where documents have a complex structure and are frequently multimodal, containing text, figures, and tables. Furthermore, we aim to outperform the different methods that Amazon Bedrock currently has to offer, showing clear signs for the need of further improvement.

3 Methodology

This section presents the methodological framework for evaluating agentic retrieval systems, split into four main research tracks: document chunking, graph-augmented retrieval, hybrid fusion, and conditional reranking. Each of these components can play a crucial role in improving the overall performance of an agent, both in terms of accuracy, as well as latency, particularly in the context of enterprise knowledge bases containing long, domain-specific documents.

3.1 Document Chunking Methods

Document chunking is often regarded as the first key step in any retrieval pipeline, since it essentially determines how long-form text is segmented into retrievable units. While there are many different chunking strategies with varying strengths and weaknesses, we have decided to focus on four distinct approaches, ranging from simple, deterministic methods to new, LLM-guided segmentation techniques.

3.1.1 Fixed-window Chunking

Fixed window chunking consists of a sliding window approach where documents are divided into chunks of predetermined size, while preserving a certain threshold of overlap. For a document D of length $|D|$ tokens (or words), we define the chunks C_i as:

$$C_i = D[i \cdot (w - o) : i \cdot (w - o) + w], \quad \text{for } i = 0, 1, \ldots, \left\lceil \frac{|D| - w}{w - o} \right\rceil \tag{1}$$

where w is the window size in tokens and o represents the overlap between consecutive chunks ($o < w$). In other words, each chunk contains exactly w tokens, and consecutive chunks share o tokens of overlapping content. While this deterministic method is computationally efficient, its main limitation lies in the fact that it has no document structure awareness, often leading to mid-paragraph splits. This can potentially degrade retrieval performance on queries requiring context on an entire section.

3.1.2 Semantic Chunking

Semantic chunking tackles the main limitations of fixed-window chunking by taking natural document boundaries such as paragraphs, sections, and subsections into consideration. The chunking procedure is composed of two steps:

1. Documents are parsed to identify structural elements (headers, paragraph breaks, etc.).

2. These elements are aggregated into chunks, capped by a maximum token threshold $T_{\max}$; if a chunk exceeds $T_{\max}$, it is subdivided at sentence boundaries using embedding-based similarity scoring.

Formally, given a sequence of sentences $S = \{s_1, s_2, \ldots, s_n\}$ within a paragraph, one approach would be to compute pairwise cosine similarities between consecutive sentence embeddings:

$$\text{sim}(s_i, s_{i+1}) = \cos(\mathbf{e}(s_i), \mathbf{e}(s_{i+1})) = \frac{\mathbf{e}(s_i) \cdot \mathbf{e}(s_{i+1})}{\|\mathbf{e}(s_i)\| \|\mathbf{e}(s_{i+1})\|} \tag{2}$$

where $\mathbf{e}(\cdot)$ denotes the vector embedding function (in our case we will be using the model Amazon Titan V2). Chunk boundaries can be identified by considering local minima of this similarity function, corresponding to semantic discontinuities. While this approach generates more coherent chunks than fixed-window chunking, it may introduce high variance in chunk sizes, which is especially an issue in technical documents with irregular section lengths. While some chunks may contain fewer than 100 tokens, others can hold up to 2'000 tokens, increasing the complexity of downstream retrieval.

3.1.3 Hierarchical Chunking

Hierarchical chunking, known as the state-of-the-art deterministic chunking method, introduces a two-level, tree-like structure, designed to find the right balance between retrieval precision and contextual coverage. This approach creates parent-child relationships between larger contextual segments and smaller, more specific segments. Throughout the chunking process, both child chunks C_{child} and parent chunks C_{parent} are generated according to predetermined token limits:

$$|C_{\text{child}}| \leq \tau_{\text{child}}, \quad |C_{\text{parent}}| \leq \tau_{\text{parent}}, \quad \text{with } \tau_{\text{child}} < \tau_{\text{parent}} \tag{3}$$

The construction process can once again be divided into two steps:

1. Documents are segmented into parent chunks based on the maximum token limit τ_{parent}
2. Each parent chunk is then further split into child chunks with maximum size τ_{child}; parent-child relationships are established by direct containment, i.e. each child chunk $C_{\text{child}}^{(j)}$ is linked to exactly one parent chunk $C_{\text{parent}}^{(i)}$ such that $C_{\text{child}}^{(j)} \subseteq C_{\text{parent}}^{(i)}$

The retrieval mechanism first identifies the most similar child chunk embeddings (e.g. using cosine similarity), as they entail more granular information, before moving on to the corresponding parent chunks to provide a broader context for answer generation. Thus, this mechanism sacrifices retrieval precision in return for contextual completeness, under the assumption that relevant information is often surrounded by supporting context within the parent chunk. A notable limitation is that the number of returned results may be fewer than requested when multiple retrieved child chunks share the same parent, potentially reducing coverage for queries requiring evidence from different sources.

3.1.4 Agentic Chunking

While each of these deterministic chunking approaches would provide us with a well-defined chunked corpus, they lack awareness of both document-level context and semantic content. This can lead to segmenting documents at positions where the different components clearly depend on each other, or making the chunk lose important general knowledge stored in other chunks linked to the same original document. Based on the idea that was first proposed by [Anthropic, 2024], we introduce two new chunking approaches, given the names "agentic chunking", that leverage LLMs to enhance chunk quality. While Anthropic proposed the conceptual framework for contextual enhancement, their approach has not been systematically evaluated against traditional chunking baselines. Furthermore, we propose a second agentic chunking method, which relies on an LLM to identify key breakpoints that are then used to split each document into chunks.

3.1.4.1 Approach 1: Contextual Enhancement The first proposed agentic approach applies contextual enhancement to chunks as post-processing. It can be regarded as an additional layer to the base chunking process (fixed, semantic, hierarchical), allowing the chunks to maintain document-level awareness. This can be done by invoking an LLM to generate a short, contextual summary of the chunk with respect to the entire document. For our analysis, we used the following prompt (adapted from [Anthropic, 2024]):

```
<document>
{document_content}
</document>
Here is the chunk we want to situate within the whole document:
<chunk>
{chunk_content}
</chunk>
Please give a short succinct context to situate this chunk within the
overall document for the purposes of improving search retrieval of the
chunk. Answer only with the succinct context and nothing else.
```

The LLM-generated context typically consists of 2-4 sentences explaining the chunk's thematic role, its relationship to surrounding content, and any relevant document-level metadata. This context is subsequently added to the respective chunk before being embedded. It is noteworthy that this method heavily relies on the initial prompt, as well as the temperature of the model, both points requiring further analysis in future work.

3.1.4.2 Approach 2: LLM-based Segmentation + Contextual Enhancement

The second agentic approach no longer builds upon any of the previously mentioned deterministic chunking methods, but rather allows an LLM to identify key breakpoints of a document in a natural, content-aware manner. Intuitively, it makes sense that an LLM would be able to recognize semantically coherent breakpoints more effectively than rule-based heuristics.

Given a document D, we first split it into sentences $S = \{s_1, \ldots, s_n\}$. We then invoke an LLM to identify key breakpoints and return them as a list of numbers, symbolizing the edges of each chunk. To avoid the LLM from returning too few or too many segments, we provide a guardrail range when it comes to the number of chunks, with the base chunking methods as fallback option if it fails to do so. For a document containing $|D|$ words, we compute a target range of $[\max(2, \tau_{\text{target}} - 3), \tau_{\text{target}} + 3]$ chunks, where $\tau_{\text{target}} = \text{round}(|D|/1000)$, reflecting a heuristic of approximately 1˙000 words per chunk (a heuristic we also applied to the previously mentioned chunking methods in our analysis). The following prompt was used to generate these breakpoints:

```
You are analyzing an academic research paper to identify optimal
breaking points for text chunking. The document has {n_sentences}
total sentences and {word_count} words. Here are the sentences:

0: {sentence_0}
1: {sentence_1}
...

Your task is to identify {min_breakpoints}-{max_breakpoints} optimal
breaking points (sentence numbers) where the document naturally
transitions between:
  - Different topics or concepts
  - Section boundaries
  - Logical flow breaks
  - Conceptual shifts
```

```
  - Changes in focus or subject matter

Return ONLY the sentence numbers as a comma-separated list (e.g.,
"8, 15, 23, 31"). Do not include explanations or any other text.
```

The model returns a list of breakpoint indices $B = \{b_1, b_2, \ldots, b_k\}$, which we then use to partition the sentence sequence into chunks: $C_1 = S[0 : b_1]$, $C_2 = S[b_1 : b_2]$, etc. After successfully having segmented the document, we apply the same contextual enhancement procedure described in Approach 1, invoking the LLM a second time to generate situating summaries for each resulting chunk. Thus, the complete pipeline involves two LLM steps per document: one for segmentation and k additional calls for contextual enrichment (where k is the number of chunks produced). While this creates additional overhead and cost, it could potentially be justified if the LLM's structural understanding yields more semantically coherent chunks than deterministic methods. Once again, it is worth noting that this approach relies heavily on the prompt that we pass to the LLM, hinting to the need of further study on this topic.

3.2 Graph-Augmented Retrieval

The previously outlined agentic approaches can be considered a first step in the right direction as far context sharing is concerned. Nevertheless, traditional vector-based retrieval treats chunks independently, lacking any awareness of relationships between entities, chunks, or documents. Graph-augmented methods aim to fix this issue by leveraging graph-implemented knowledge bases that store semantic and structural connections, making it particularly useful for retrieval that requires chunks from multiple different documents.

3.2.1 Graph Construction

The GraphRAG framework, originally proposed in [Edge et al., 2025] and summarized in Figure 3, begins by extracting entities and relationships from document chunks using an LLM. For each chunk C_i, the LLM identifies named entities $E_i = \{e_1, e_2, \ldots, e_m\}$ (people, organizations, locations, concepts) and pairwise relationships $R_i = \{(e_a, r, e_b)\}$ that appear within the chunk. When repeating this action across all chunks, we are able to set up a knowledge graph $G = (V, E)$. The graph consists of two types of nodes and two types of edges:

- Nodes (V): Entity nodes $e \in E_{\text{entities}}$ and chunk nodes $c \in C_{\text{chunks}}$
- Edges (E): Semantic relationships (e_i, r, e_j) between entities and containment relationships (e, c) indicating entity e appears in chunk c

This bipartite graph structure allows for traversal from query-relevant entities to the chunks that entail them, as well as exploration of multi-hop entity chains. Moreover, GraphRAG uses the Leiden algorithm, first introduced in [Traag et al., 2019], adopting a community detection approach to partition V into densely connected communities $\{G_1, G_2, \ldots, G_k\}$ by optimizing the following modularity:

$$Q = \frac{1}{2m} \sum_{i,j} \left[A_{ij} - \frac{k_i k_j}{2m} \right] \delta(c_i, c_j) \tag{4}$$

where A_{ij} is the adjacency matrix, k_i is the degree of node i, m is the total number of edges, c_i denotes the community assignment of node i, and $\delta(\cdot, \cdot)$ is the Kronecker delta. For each detected community, the LLM generates a high-level summary describing the global topic and key entities within that subgraph, enabling top-down retrieval.

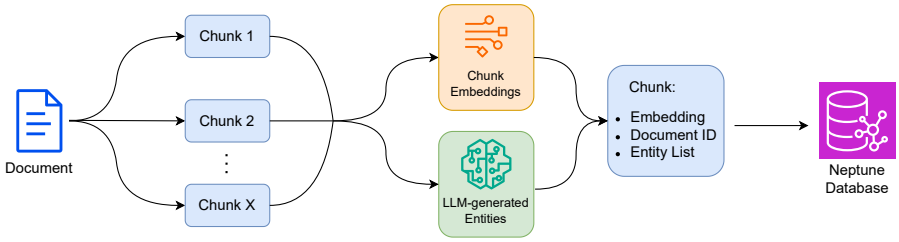

Figure 3: Summarized overview of our GraphRAG construction pipeline.

3.2.2 Graph Traversal & Retrieval

GraphRAG supports two different retrieval methods: local search and global search, depending on the granularity of the query. While local search focuses on targeting entity-centric subgraphs (i.e. subgraphs where single entity nodes share many edges with other chunk/entity nodes), global search leverages community summaries when facing broader questions. In our work, we focus primarily on local search, as our datasets were designed to target very specific knowledge retrieval, whether it be single-hop (involving only one chunk) or multi-hop (involving several chunks).

Given a query q, local search first uses an LLM to extract key entities E_q from q, by using the exact same extraction prompt applied during the graph construction. These extracted entities serve as a direct entry point into the knowledge graph, as we then retrieve all chunks containing any of these seed entities:

$$C_{\text{0-hop}} = \{c \in C_{\text{chunks}} \mid \exists e \in E_q \text{ such that } (e, c) \in E\} \tag{5}$$

Next, to enable multi-hop reasoning, we expand the search by traversing entity-entity relationship edges. Here, we identify both 1-hop and 2-hop neighboring entities:

$$E_{\text{1-hop}} = \{e' \in E_{\text{entities}} \mid \exists e \in E_q, (e, r, e') \in R_{\text{entity-entity}}\} \tag{6}$$
$$E_{\text{2-hop}} = \{e'' \in E_{\text{entities}} \mid \exists e' \in E_{\text{1-hop}}, (e', r, e'') \in R_{\text{entity-entity}}\} \tag{7}$$

We then collect all chunks associated with these expanded entity sets:

$$C_{\text{1-hop}} = \{c \in C_{\text{chunks}} \mid \exists e \in E_{\text{1-hop}}, (e, c) \in E\} \tag{8}$$
$$C_{\text{2-hop}} = \{c \in C_{\text{chunks}} \mid \exists e \in E_{\text{2-hop}}, (e, c) \in E\} \tag{9}$$

Finally, the candidate pool targeted for retrieval is simply the union of all retrieved chunks:

$$C_{\text{local}} = C_{\text{0-hop}} \cup C_{\text{1-hop}} \cup C_{\text{2-hop}} \tag{10}$$

These candidate chunks are then re-ranked by computing their semantic similarity with respect to the query, before returning the top-k chunks. This expansion schema allows

to retrieve evidence multi-hop reasoning, e.g. when answering "What is the relationship between entity A and entity C?", where only direct relationships exist between A→B and B→C, the 2-hop expansion can discover entity C through the intermediate entity B, allowing the system to connect information from multiple different sources.

To guarantee a fair comparison, we evaluate GraphRAG against the best-performing non-graph baseline from Section 3.1, on both single-hop and multi-hop questions. This will be one of the very first apples-to-apples comparisons between traditional chunking and embedding methods, and graph-structured knowledge bases.

3.3 Hybrid Retrieval

Modern retrieval systems often rely purely on semantic search and dense vector embeddings to identify relevant passages [Karpukhin et al., 2020]. While dense retrievers thrive at semantic similarity, they often struggle at identifying keyword matches, numerical values, or domain-specific terminology, as shown in [Thakur et al., 2021]. Hybrid retrieval addresses these limitations by fusing sparse lexical methods with dense semantic representations, leveraging their complementary strengths.

A major challenge in hybrid retrieval lies in determining the optimal balance between sparse and dense signals. Traditional approaches test for several fusion weights, before doubling down on one single weight [Bruch et al., 2023]. This makes the entire process static, despite the ideal balance varying substantially depending on query characteristics: short, keyword-heavy queries benefit from sparse retrieval, while semantically complex queries require rely heavily on dense retrieval. To tackle this limitation, we aim to learn query-specific weights that adapt the fusion dynamically based on individual query features.

3.3.1 Sparse Retrieval: BM25

BM25 (Best Match 25), first introduced in [Robertson and Zaragoza, 2009], remains one of the state-of-the-art lexical retrieval methods, due to its balance between simplicity and efficiency. Given a query q and document d, BM25 computes a relevance score by aggregating term-level contributions:

$$\text{BM25}(q, d) = \sum_{i=1}^{m} \text{IDF}(q_i) \cdot \frac{f(q_i, d) \cdot (k + 1)}{f(q_i, d) + k \cdot (1 - b + b \cdot |d|/\text{avg}(|d|))} \tag{11}$$

Here, $f(q_i, d)$ denotes the frequency of term q_i in document d, $|d|$ is the document length in tokens, and avg($|d|$) is the average document length across the corpus. The inverse document frequency (IDF) factor reduces the weight of common terms, having the following formulation:

$$\text{IDF}(q_i) = \log \left[\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1 \right] \tag{12}$$

where N is the total number of documents and $n(q_i)$ is the document frequency of term q_i. Meanwhile, k and b can be seen as tuning parameters that control term frequency saturation and length normalization. Following standard practice [Robertson and Zaragoza, 2009], we set $k = 1.5$ and $b = 0.75$, which have proven to be robust across diverse document collections. BM25 can be regarded as an extension of the classical NLP

algorithm Term Frequency-Inverse Document Frequency (TF-IDF), however, introducing a non-linear saturation of term frequencies.

3.3.2 Dense Retrieval: Semantic Similarity

For dense retrieval, chunks are turned into embeddings via a transformer-based encoder. In our case, we will use Amazon Titan V2, generating vectors of dimension 1˙024. These embeddings are learned in a way such that texts similar meaning are mapped close to each other in the embedding space, enabling retrieval based on semantic similarity rather than exact keyword matching.

Given query q and document d, embeddings $\mathbf{e}_q$ and $\mathbf{e}_d$ are computed independently, and relevance can be measured via a similarity metric, e.g. the cosine similarity.

Dense retrieval performs particularly well when queries and documents exhibit lexical variation but share similar context. However, this semantic comparison also has its downsides, as embeddings may confuse similar concepts (e.g. different numerical values), leading to false positives when precision is critical.

3.3.3 Query-aware Hybrid Fusion

3.3.3.1 Fusion Mechanism & Normalization The first step of hybrid fusion consists of retrieving a certain number of chunks k (we take $k = 10$) from both BM25 and semantic search based on their respective scores. By taking the union of both lists of retrieved chunks, we then get a set of up to $2k$ distinct chunks. Next, the goal is to combine the computed scores into a combined relevance score, leveraging the complementary strength of both methods. The scores can be fused in the following way, using a query-specific weight $\alpha(q) \in [0, 1]$:

$$R_{\text{hybrid}}(c) = \alpha(q) \cdot s_{\text{sem}}(q, c) + (1 - \alpha(q)) \cdot s_{\text{BM25}}(q, c) \tag{13}$$

where q is the query and c the chunk. However, the scores outputted by both approaches are not on the same scale. BM25 score are positive, unbounded and depend on query length and term statistics, while cosine similarity ranges between [-1,1]. Direct combination of both scores would lead to unstable results, highlighting the need for normalization first. To tackle this issue, we adopt theoretical min-max normalization (TM2C2), the approach shown to be the most robust in terms of data distribution according to [Bruch et al., 2023]. This method scales scores to [0,1] using the theoretical minimum of each scoring function:

$$\tilde{s}_{\text{sem}}(q, c) = \frac{\text{sim}_{\text{sem}}(q, c) - \inf(\text{sim}_{\text{sem}})}{\max_{c' \in \mathcal{C}_q} \text{sim}_{\text{sem}}(q, c') - \inf(\text{sim}_{\text{sem}})} \tag{14}$$

$$\tilde{s}_{\text{BM25}}(q, c) = \frac{\text{BM25}(q, c) - \inf(\text{BM25})}{\max_{c' \in \mathcal{C}_q} \text{BM25}(q, c') - \inf(\text{BM25})} \tag{15}$$

This approach is shown to be more robus than regular min-max normalization, as it is prone to outliers in the retrieved set and ensures consistent scaling across queries. Thus, hybrid fusion becomes:

$$R_{\text{hybrid}}(q, c) = \alpha(q) \cdot \tilde{s}_{\text{sem}}(q, c) + (1 - \alpha(q)) \cdot \tilde{s}_{\text{BM25}}(q, c) \tag{16}$$

Unlike traditional fixed-weight approaches where α is kept constant for all queries, our adaptive mechanism learns to predict $\alpha(q)$ based on query features, allowing the system to naturally give more importance to the more relevant retrieval strategy.

3.3.3.2 LSS & Optimal α Computation In order to train a query-specific weight predictor, we require a metric that ranges from [-1,1] and that is more or less consistent with the relevance scoring. As a proxy for chunk relevance, we opted for the Longest Subsequence Similarity (LSS), defined as follows:

$$\text{LSS}(c, a) = \frac{|\text{LCS}(\text{words}(c), \text{words}(a))|}{|\text{words}(a)|} \tag{17}$$

where c is a chunk, a the ground-truth answer (which is an exact substring of the "correct" chunk in our dataset), $\text{LCS}(\cdot, \cdot)$ computes the longest common subsequence, and $\text{words}(\cdot)$ tokenizes text into words. In simpler terms, LSS measures the proportion of answer words that appear, in order, in the candidate chunk c. LSS is bounded in $[0, 1]$, with $\text{LSS} = 1$ indicating that all answer words appear in sequence in the document. We choose this as a proxy, since we'd like to learn $\alpha(q)$ so that high LSS scores and high fused are more or less synonymous.

In practice, we compute the LSS for each chunk that is in either the top-k BM25 or semantic chunks. This is done for each query q in our QA dataset, with respect to the ground-truth answer a_q (which is available in our dataset). Our objective is to find the "optimal" weight $\alpha^*(q)$ that minimizes the mean squared error (MSE) between the fused score and the LSS label:

$$\alpha^*(q) = \arg\min_{\alpha \in [0,1]} \sum_{d \in \mathcal{D}_q} \left(\text{LSS}(d, a_q) - [\alpha \cdot \tilde{s}_{\text{sem}}(q, d) + (1 - \alpha) \cdot \tilde{s}_{\text{BM25}}(q, d)]\right)^2 \tag{18}$$

Since this is a convex quadratic optimization problem, admitting a closed-form solution, we can simply take its derivative with respect to α and set it to 0, giving us:

$$\alpha^*(q) = \frac{\sum_{d \in \mathcal{D}_q} \left(\text{LSS}(d, a_q) - \tilde{s}_{\text{BM25}}(q, d)\right) \cdot \left(\tilde{s}_{\text{sem}}(q, d) - \tilde{s}_{\text{BM25}}(q, d)\right)}{\sum_{d \in \mathcal{D}_q} \left(\tilde{s}_{\text{sem}}(q, d) - \tilde{s}_{\text{BM25}}(q, d)\right)^2} \tag{19}$$

$\alpha^*(q)$ is computed for each query in our training set, providing ground-truth targets for supervised learning.

3.3.3.3 Learning to Predict α We train three distinct models of increasing complexity to predict $\alpha(q)$ based on query features, while considering the trade-off between accuracy and computational efficiency.

Ridge Regression. One of the simplest ways is to use a linear model with four easily extractable and interpretable query features: (1) word count, (2) has digits, (3) has wh-question words (who, what, where, when, why, how), (4) has acronyms (contains uppercase sequences). We fit a ridge regression model with L_2 regularization:

$$\hat{\alpha}(q) = \mathbf{w}^T \mathbf{x}(q) + b \tag{20}$$

where $\mathbf{x}(q) \in \mathbb{R}^4$ is the feature vector. The feature weights $\mathbf{w} \in \mathbb{R}^4$ and bias $b \in \mathbb{R}$ are learned by minimizing the regularized mean squared error:

$$\mathbf{w}^*, b^* = \arg\min_{\mathbf{w},b} \sum_{q \in \mathcal{Q}_{\text{train}}} \left(\alpha^*(q) - (\mathbf{w}^T \mathbf{x}(q) + b)\right)^2 + \lambda \|\mathbf{w}\|^2 \tag{21}$$

where λ is the regularization parameter tuned via cross-validation. Ridge regression admits a closed-form solution obtained via matrix operations. Given the design matrix $\mathbf{X} \in \mathbb{R}^{n \times 4}$ (where row i contains features for query q_i) and target vector $\mathbf{y} \in \mathbb{R}^n$ (where $y_i = \alpha^*(q_i)$), the optimal weights are:

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y} \tag{22}$$

This closed-form solution allows for direct resolution of the problem, while being highly interpretable, since the weights provide us with the directional importance of each feature. However, we are making a big assumption by only considering a linear combination, something that might restrict the model from learning complex query-level patterns.

XGBoost. To tackle the linear limitation of the previous method and incorporate richer information, we use XGBoost (eXtreme Gradient Boosting) [Chen and Guestrin, 2016], an ensemble method that iteratively builds a sequence of decision trees, where each tree corrects the errors of the previous ones through gradient-based optimization. Moreover, we increase the set of features to also include the normalized relevance scores of the retrieved chunks from both BM25 and semantic search. Thus, we are left with 25 features in total: (1) 4 simple query features used previously, (2) top-10 normalized semantic scores: $\{\tilde{s}_{\text{sem}}(q, c_1), \ldots, \tilde{s}_{\text{sem}}(q, c_{10})\}$, (3) top-10 normalized BM25 scores: $\{\tilde{s}_{\text{BM25}}(q, c_1), \ldots, \tilde{s}_{\text{BM25}}(q, c_{10})\}$, (4) overlap count: $|\mathcal{C}_{\text{sem}}(q) \cap \mathcal{C}_{\text{BM25}}(q)|$. The model learns a non-linear ensemble of decision trees:

$$\hat{\alpha}(q) = \sum_{t=1}^{T} f_t(\mathbf{x}(q)) \tag{23}$$

where each f_t is a regression tree and T is the number of boosting rounds. Unlike ridge regression, XGBoost is able to capture complex, non-linear relationships between features. Overall, this method offers a good balance between interpretability and efficiency, requiring moderate computational resources.

Multilayer Perceptron (MLP). Rather than relying on extracted or score-based features, we can also consider a deep learning approach, using a multilayer perceptron (MLP) that operates on the query embeddings, outputting the predicted scalar $\hat{\alpha}(q)$. In our case, we consider a lightweight architecture, consisting of three fully connected layers with ReLU activations:

$$\mathbf{h}_1 = \text{ReLU}(\mathbf{W}_1 \mathbf{e}_q + \mathbf{b}_1), \quad \mathbf{h}_1 \in \mathbb{R}^{n_1} \tag{24}$$

$$\mathbf{h}_2 = \text{ReLU}(\mathbf{W}_2 \mathbf{h}_1 + \mathbf{b}_2), \quad \mathbf{h}_2 \in \mathbb{R}^{n_2} \tag{25}$$

$$\hat{\alpha}(q) = \sigma(\mathbf{W}_3 \mathbf{h}_2 + \mathbf{b}_3) \tag{26}$$

where $\sigma(\cdot)$ is the sigmoid function ensuring that $\hat{\alpha}(q) \in [0, 1]$. Here, we use the AdamW optimizer [Loshchilov and Hutter, 2019], while testing for different layer widths (n_1 and n_2), learning rates, and dropout rates. We then train our model with respect to the MSE loss:

$$\mathcal{L} = \frac{1}{|\mathcal{Q}_{\text{train}}|} \sum_{q \in \mathcal{Q}_{\text{train}}} (\alpha^*(q) - \hat{\alpha}(q))^2 \tag{27}$$

The strength of the MLP is that it can learn highly non-linear mappings from the embeddings, making it a strong candidate to capture complex semantical patterns. Nevertheless, despite only being a lightweight MLP, this expressiveness comes at a high computational cost compared to the simpler baselines.

3.4 Reranking Methods

While hybrid retrieval is one way to potentially improve the quality of the retrieved candidate chunks, the top-ranked chunks may still be ranked suboptimally due to limitations in scoring functions. A solution for this is to rerank the retrieved chunks by applying a method that jointly considers the query and the list of candidate chunks, refining the initial ranking. However, advanced rerankers, such as those that rely on LLMs, introduce significant cost and latency, motivating the need for a conditional reranking approach, where the procedure is only triggered if the scores are below a certain threshold.

3.4.1 LLM-based Reranking

Recent work has shown that LLMs can act as effective rerankers when given the right zero-shot prompt [Sun et al., 2024]. Here, the model's reasoning capabilities are used to assess chunk relevance. Given a query q and a list of top-k candidate chunks $\{c_1, c_2, \ldots, c_k\}$ (ordered by hybrid fusion score), we create a prompt that combines the query and all candidates, instructing the model to rerank them by relevance:

```
Given the following query and retrieved chunks (ordered by highest
score to lowest), rank the chunks by their relevance to answering
the query.

Query: {question}

Chunks:{chunks_text}

Please analyze each chunk and return ONLY a JSON array of document
numbers ranked from most relevant to least relevant.
Example format: [1, 5, 3, 7, 2, ...]

Return ONLY the JSON array, nothing else.
```

The LLM should then theoretically return a permutation of document indices, representing the reranked order.

3.4.2 Conditional Reranking

The main benefit of considering a conditional reranking framework is because not all queries benefit equally from expensive reranking. In the case where the initial retrieval shows high-confidence fused scores, the ranking tends be more or less accurate, such that reranking would only yield minimal improvement. Conversely, chunks with low-confidence scores tend to be more ambiguous with respect to the query, making them ideal candidates for reranking.

3.4.2.1 Decision Rule Let us define a confidence threshold $\tau \in [0, 1]$, a threshold that we will compare to the maximum fusion score for each query. For a query q, let $s_{\max}(q) = \max_{c \in \mathcal{C}_q} R_{\text{hybrid}}(q, c)$ be the highest fusion score among retrieved candidates. The reranking decision is:

$$\text{rerank}(q) = \begin{cases} \text{True} & \text{if } s_{\max}(q) < \tau \\ \text{False} & \text{otherwise} \end{cases} \tag{28}$$

When $\text{rerank}(q) = \text{True}$, we apply LLM-based reranking to all retrieved chunks, otherwise, we return the documents following their original ranking. The final top-k documents are:

$$\text{top-}k(q) = \begin{cases} \text{top-}k \text{ by } r_{\text{LLM}}(c) & \text{if rerank}(q) = \text{True} \\ \text{top-}k \text{ by } R_{\text{hybrid}}(q, c) & \text{otherwise} \end{cases} \tag{29}$$

where $r_{\text{LLM}}(c)$ denotes the LLM-assigned rank.

3.4.2.2 Threshold Calibration via Wilson Score Interval The main challenge lies in selecting τ^* to ensure that skipped queries (those with $s_{\max}(q) \geq \tau$) maintain a high precision. For this, we use the Wilson Score Interval [Wilson, 1927], a statistical method for calculating the confidence interval of a binomial proportion, to calibrate τ with a certain confidence-level. A binomial proportion is defined as the ratio of "successes" to the total number of trials in an experiment that only has two possible outcomes.

Let τ be the candidate threshold, let $n(\tau)$ be the number of queries in the training/-validation set satisfying $s_{\max}(q) \geq \tau$ (these are the queries that would be "skipped", i.e. that would not be reranked), and let $k(\tau)$ denote the number of these queries where at least one of the top-5 chunks (based on the fusion score) contains the exact ground-truth answer. This would give us an empirical precision of:

$$\hat{p}(\tau) = k(\tau)/n(\tau) \tag{30}$$

The Wilson score 95% confidence interval provides a lower bound on the true precision:

$$\text{CI}_{\text{lower}}(\tau) = \frac{1}{1 + \frac{z^2}{n}} \left(\hat{p} + \frac{z^2}{2n} - z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}} \right) \tag{31}$$

where $z = 1.96$ for 95% confidence. We then select the optimal threshold as:

$$\tau^* = \min \{\tau \in [0, 1] : \text{CI}_{\text{lower}}(\tau) \geq \beta \text{ and } n(\tau) \geq 30\} \tag{32}$$

where β is the desired minimum precision. This approach guarantees two things:

1. Skipped queries have at least β precision with 95% statistical confidence
2. The skip group contains at least 30 queries, providing sufficient sample size for reliable estimation of the confidence interval

By choosing the minimum τ, we maximize the skip rate, i.e. minimize the reranking cost. Furthermore, the reranking threshold τ^* can be recalibrated as new data becomes available.

4 Experimental Setup & Data

This section describes the experimental framework for evaluating the agentic retrieval methods introduced in Section 3. We present the datasets and corpora used for evaluation, detail the implementation of each method with their respective configurations, and specify the evaluation metrics and baseline comparisons that allow us to rigorously measure the retrieval accuracy, efficiency, and cost trade-offs.

4.1 Datasets

To avoid any confidentiality issues of using Amazon internal data, we leveraged open-source resources to come up with meaningful datasets for our experiments. While there are several high-quality, open-source benchmark datasets available to evaluate retrieval systems, most of them lack the similarity to what can be found in long-form, complex-structured enterprise documents. Therefore, we chose to generate our very own evaluation datasets. For this, we decided that academic research papers could form a solid, representative foundation to test our knowledge base, as they are typically multimodal files (containing figures and tables) with diverse formatting. More specifically, we selected all publicly available Amazon research publications (can be found here: `https://www.amazon.science/publications`), published between January 1, 2024 and September 14, 2025, which amounted to 982 documents in total. It is worth noting that we retrieved the PDF for each of these papers, alongside the tags (e.g. computer vision) that had been assigned to them.

The next step consisted of transforming these files into markdown (md), a more "digestible" format for retrieval. This conversion is far from straightforward, since PDF files are stored in binary and manual processing is not feasible for such a large corpus. Therefore, we chose Docling [Auer et al., 2024], a state-of-the-art PDF to md parser developed by IBM. The model combines optical character recognition (OCR) and natural language processing (NLP) to detect and segment document elements (like paragraphs, figures, tables), before recombining them into JSON format that preserves both content and structure. For each paper, we only kept the first pages and disregarded any files of size greater than 10MB.

Next, we split these converted md files into segments of 1˙000 words, discarding any that had a word count below 500. Based on these segments, we generated three types of questions and answers (QAs):

1. Single-file 1-hop (SF1H): QA related to one single segment in a file
2. Single-file 2-hop (SF2H): QA related to two distinct segments from the same file
3. Multi-file 2-hop (MF2H): QA related to two distinct segments from two distinct files

All of these QAs were generated by invoking an LLM (in our case Claude Opus 4.1) with clear and well-defined prompts. For SF1H QAs, we asked the LLM to produce between 1-3 extractive QAs regarding that specific segment, with the ground-truth answer necessarily being an exact substring of the passage. For both 2-hop QA datasets (SF2H and MF2H), we started off by performing Named Entity Recognition (NER) (using the Python library spaCy [Honnibal et al., 2020], focusing primarily on entities of the type "PERSON", "ORG", "PRODUCT", and "WORK_OF_ART") to extract key entities from

each segment. If the number of obtained entities for a particular segment was below five, we then asked an LLM to generate the remaining ones to complete the list, so that we had exactly 5 entities per segment. Next, we decided to create pairs of distinct segments sharing the same entities, SF2H only considering same file pairs and MF2H only considering cross-file pairs. To avoid segments from two completely different research fields being bundled up, we only allowed connections between those for which the original research papers shared at least one tag (which we stored from the beginning). After having established a list of well-defined pairs, we then tasked an LLM with generating 1-3 QAs, emphasizing on the fact that the questions should require both passages to be able to respond correctly. Once again, the answer would be an exact substring of one of the targeted segments. In total, this lead to having the following number of QAs: 19˙389 in SF1H, 13˙691 in SF2H, and 1˙567 in MF2H.

Overall, this approach allows us to test both single-hop retrieval and multi-hop reasoning capabilities, providing a fair and comprehensive evaluation framework that reflects the complexity of real-world queries in enterprise knowledge bases. The code for the dataset generation, alongside the three datasets themselves, stored in JSON format, can be found linked in the Appendix A.

4.2 Implementation

All of the conducted experiments were implemented using Python 3.12.3, leveraging AWS infrastructure and API resources. This subsection outlines the implementation of each of the previously discussed retrieval methods.

For the three basic chunking methods (fixed-window, semantic, hierarchical) we developed custom Python code, deployed on AWS Lambda functions to perform document transformations at scale. For agentic chunking, we additionally invoke Claude Haiku 3.5 via the AWS Bedrock API for both contextual enhancement and LLM-driven segmentation. We set the temperature to 0.2 to favor consistent outputs and limit the maximum number of output tokens to 1˙000. In the case where a token limit is hit for Claude Haiku 3.5, we use Claude Haiku 3 as a fallback option, allowing for up to a maximum of 3 retry attempts. For the basic chunking methods, we perform grid search over 6 different configurations per chunking method:

- Fixed-window: (`maxTokens`, `overlapPercentage`)$\in \{(700, 10), (700, 20), (1˙000, 10), (1˙000, 20), (1˙300, 10), (1˙300, 20)\}$
- Semantic: (`breakpointPercentileThreshold`, `bufferSize`, `maxTokens`)$\in \{(80, 0, 800), (80, 1, 800), (80, 0, 1˙000), (80, 1, 1˙000), (90, 0, 1˙200), (90, 1, 1˙200)\}$
- Hierarchical: (`parentTokens`, `childTokens`, `overlapTokens`)$\in \{(2˙000, 500, 50), (2˙000, 500, 100), (2˙000, 800, 50), (2˙000, 800, 100), (3˙000, 1˙000, 50), (3˙000, 1˙000, 100)\}$

For graph-augmented retrieval, we replicate the GraphRAG methodology [Edge et al., 2025] using AWS infrastructure, combining Amazon Bedrock Knowledge Bases for entity extraction and Amazon Neptune for graph storage and traversal. Entity extraction is performed using Claude Haiku 3.5 with temperature 0.2 and maximum output tokens set to 1000. The extraction prompt requests entities in structured JSON format with fields for entity text, type, and relationships. Extracted entities are then stored, alongside their relationships, in Amazon Neptune, a fully managed graph database service optimized for

storing and querying highly connected data. For the remaining hyperparameters, we use the default GraphRAG configurations, following the original Microsoft implementation adapted to AWS.

For sparse retrieval, we use the library `bm25_rank` [Brown, 2020], a Python implementation of the BM25 algorithm. Retrieval scores and document rankings are computed locally and stored in memory. For dense retrieval, we generate embeddings using Amazon Titan V2 via the AWS Bedrock API. The 1024-dimensional vector embeddings are stored in OpenSearch Service, which uses cosine similarity as the distance metric.

We implement three models to predict the query-specific weight $\alpha(q)$ in hybrid fusion: ridge regression, XGBoost, and MLP. Ridge regression is implemented using `scikit-learn`'s `Ridge` class, with L_2 regularization strength λ selected via 5-fold cross-validation. XGBoost is implemented using the official `xgboost` Python library, with hyperparameters selected via 5-fold cross-validation on the training set. The MLP is implemented using `PyTorch` and consists of an input layer (1024 dimensions), two hidden layers with ReLU activations and dropout, and a sigmoid output layer returning $\hat{\alpha}(q) \in [0, 1]$. For training, we use the AdamW optimizer with the MSE loss, allowing early stopping with patience of 10 epochs and a maximum of 100 epochs. All three models are trained on a 70/15/15 train/validation/test split of the combined QA dataset (SF1H + SF2H + MF2H), using a stratified split with random seed 42 to maintain proportional representation of each dataset. Hyperparameters are selected based on validation set performance, while final evaluation metrics are reported on the test set. Ridge Regression is tested on 5 configurations with $\lambda \in$ [0.01, 0.1, 1.0, 10.0, 100.0]. XGBoost is tested on 54 configurations combining `n_estimators` $\in$ [50, 100, 200], `max_depth` $\in$ [3, 5, 7], `learning_rate` $\in$ [0.01, 0.1, 0.3], and `subsample` $\in$ [0.8, 1.0]. The MLP is tested on 90 configurations combining `hidden_dims` $\in$ [(32, 16), (64, 32), (128, 64)], `learning_rate` $\in$ [0.001, 0.0001], `batch_size` $\in$ [32, 64, 128], `dropout` $\in$ [0.5, 0.7, 0.8], and `weight_decay` $\in$ [0.01, 0.001].

For LLM-based reranking, we use Claude Haiku 3.5 via the Bedrock API with temperature 0.2 and maximum output tokens set to 1000. Moreover, we implement basic error handling to deal with cases where the LLM does not output a valid JSON, falling back to the original fusion-based ranking when parsing fails.

The number of retrieved chunks depends on the considered experiment. For chunking experiments (Section 3.1) and graph-augmented retrieval experiments (Section 3.2), we retrieve the top-5 chunks from each method. For hybrid retrieval experiments (Section 3.3) and reranking experiments (Section 3.4), we retrieve the top-10 chunks from both BM25 and semantic search independently, forming union sets of up to 20 unique candidate chunks.

4.3 Evaluation

When it comes to evaluation, we consider two types of retrieval metrics, both self-implemented in Python: gold-span accuracy and chunk accuracy. Gold-span metrics measure the retrieval quality based on whether the ground-truth answer is contained in the retrieved chunk(s). We consider the following gold-span metrics:

- Top-1 Exact Match (T1EM): Binary indicator of whether the top-ranked chunk contains the exact answer substring.
- Top-5 Exact Match (T5EM): Binary indicator of whether any of the 5 top-ranked chunk contains the exact answer substring.

- Top-1 Coverage (T1C): Proportion of answer words that appear in sequence in the top-ranked chunk, computed via longest common subsequence matching.
- Best Coverage (BC): Maximum answer coverage with respect to any retrieved chunk.
- Average Coverage (AC): Mean answer coverage across all retrieved chunks per query.
- Multi-chunk Coverage (MC): Answer coverage when concatenating all retrieved chunks.
- Mean Reciprocal Rank (MRR): Average of $1/r$ where r is the rank of the first chunk containing the answer.
- Average Top-1 Score (AT1S): Average retrieval score of the top-ranked chunk.

Chunk accuracy metrics evaluate retrieval based on the overlap of retrieved chunks with the original segments used to generate the QAs. We consider the following chunk accuracy metrics:

- Top-1 Overlap (T1O): Text overlap (measured via word-level sequence matching) between the top-ranked retrieved chunk and any ground-truth segment.
- Best Overlap (BO): Maximum overlap between any retrieved chunk and any ground-truth segment.
- Average Overlap (AO): Average overlap across all retrieved chunks and all ground-truth segments.

Finally, we also measure efficiency metrics, such as cost and latency. Financial cost can be linked to the used hardware and number of API calls. In terms of latency, it is of interest to measure the average retrieval time per query by testing the entire dataset with warm-start conditions (models and indexes are preloaded). Our evaluation logic builds up progressively, where each experiment uses the best-performing method from the previous section as a baseline.

4.4 Computational Resources

All experiments were run on an AWS EC2 g5.xlarge instance, which provides 4 vCPUs (AMD EPYC 7R32), 16 GB RAM, and 1 NVIDIA A10G Tensor Core GPU with 24 GB GPU memory. Furthermore, we used the following AWS services: Amazon Bedrock (for LLM API access and Titan V2 embeddings), Amazon OpenSearch Service (for vector storage and similarity search), Amazon Neptune (for graph storage and traversal), AWS Lambda (for distributed document processing), Amazon EC2 (for experiment orchestration), and Amazon SageMaker (for hyperparameter tuning).

5 Results

5.1 Chunking Results

To come up with the optimal chunking method, we compared three deterministic approaches, fixed-window, semantic, and hierarchical, each of them tested across six hyperparameter configurations. The configurations presented in Table 1 represent the best-performing setup for each of these chunking methods based on four metrics that we identify

to be the most important throughout our entire analysis: Top-1 Exact Match (T1EM), Top-1 Coverage (T1C), Top-1 Overlap (T1O), and Mean Reciprocal Rank (MRR). These metrics value the accuracy of the top-ranked chunk, as well as how well our retrieval system does at positioning the most relevant chunk.

	SF1H				SF2H				MF2H			
Configuration	T1EM	T1C	T1O	MRR	T1EM	T1C	T1O	MRR	T1EM	T1C	T1O	MRR
fixed-700-20	0.272	0.389	0.168	0.357	0.364	0.409	0.225	0.471	0.311	0.357	0.202	0.407
semantic-80-1-800	0.324	0.430	0.121	0.408	0.362	0.405	0.150	0.470	0.288	0.333	0.148	0.375
hierarchical-2000-500-100	**0.387**	**0.507**	**0.243**	**0.482**	0.467	**0.517**	**0.318**	0.578	0.369	0.436	**0.282**	0.483
agentic-1	0.369	0.493	0.238	0.464	**0.469**	0.509	0.315	**0.581**	**0.394**	**0.450**	0.279	**0.505**
agentic-2	0.248	0.407	0.165	0.329	0.413	0.475	0.263	0.516	0.305	0.386	0.216	0.411

Table 1: Chunking performance comparison across the three datasets. The table shows the best-performing configuration for each deterministic method and both agentic approaches. Metrics include Top-1 Exact Match (T1EM), Top-1 Coverage (T1C), Top-1 Overlap (T1O), and Mean Reciprocal Rank (MRR). Bold values indicate best performance per dataset-metric combination.

When focusing only on the classical chunking methods, hierarchical outperforms the other approaches across all metrics and datasets. This validates the main idea behind this method, which looks to balance granular retrieval (child chunks) with sufficient context (parent chunks). Given this result, we used this exact hierarchical configuration as the base layer for the first agentic approach (agentic-1), which enriches the chunks with LLM-generated, document-level contextual summaries. On the other hand, the second agentic approach (agentic-2) does not rely on any form of deterministic chunking, asking an LLM to come up with both the ideal breakpoints, as well as enhancing it with further context.

The results shown in Table 1 reveal that there is no clear winner across all metrics and datasets. For SF1H, where QAs are based on information from only one segment, hierarchical returns the best results across all metrics, showcasing a T1EM of 38.7%, 4.7% better than the second-placed agentic-1 approach. Meanwhile, agentic-2 performs substantially worse with a T1EM of 24.8%, falling below even fixed-window chunking. This degradation could be correlated with the guardrails we imposed on the chunk length (targeting approximately 1000 words per chunk), forcing the LLM to make suboptimal segmentation decisions.

Meanwhile, for multi-hop questions the trend seems to reverse, with agentic-1 showing the strongest results. On both SF2H and MF2H, agentic-1 achieves the highest T1EM with 46.9% and 39.4% respectively, surpassing hierarchical by 0.4% on SF2H and 6.8% on MF2H. In particular, the additional context contributes to a significant improvement for cross-file queries (MF2H), where these summaries help the retriever understand how chunks are related to wider themes and topics. Moreover, agentic-2 also shows stronger multi-hop performance than on single-hop queries, ranking third overall across most metrics, with T1EM scores of 41.3% (SF2H) and 30.5% (MF2H). Overall, this suggests that contextual summaries becomes particularly beneficial when dealing with multi-hop reasoning, allowing to cross-reference related entities and concepts. Furthermore, when analyzing the metrics across all three datasets, we notice that SF2H consistently achieves the highest scores for all methods. This is most likely due to two-hop questions connecting specific entities across chunks, resulting in more constrained, keyword-specific queries that provide stronger retrieval signals than broader single-hop questions. Despite MF2H questions having similar specificity, SF2H outperforms MF2H because same-file chunks

share consistent vocabulary and semantic context, whereas cross-file retrieval must connect evidence from files with varying terminology and writing styles.

Configuration	Ingestion Time (s/doc)	Preprocessing Cost ($/doc)	Query Latency (s/query)
fixed-700-20	**0.445**	**0.00**	0.198
semantic-80-1-800	4.263	**0.00**	0.197
hierarchical-2000-500-100	0.573	**0.00**	0.200
agentic-1	3.822	2.45	**0.190**
agentic-2	3.881	0.13	0.236

Table 2: Summary of cost-efficiency trade-off per chunking configuration.

While agentic-1 delivers the best multi-hop performance, this improvement comes at a cost, summarized in Table 2. The method incurs a preprocessing cost of around $2.45 per document, which is based on the logic that Claude Haiku 3.5 has a cost of $0.0008 per 1˙000 input tokens and $0.004 per 1˙000 output tokens. This is significantly higher than the deterministic methods, as well as the agentic-2 approach, which produces close to 20 times fewer chunks than agentic-1 on average. In terms of ingestion time (i.e. the time it takes to chunk and perform other transformations) is fastest for fixed-window (0.445 s/doc) and hierarchical chunking (0.573 s/doc), while semantic and agentic approaches require close to 10 times longer, due to embedding computations (semantic) or LLM invocations (agentic). Interestingly, the query latency remains more or less stable across all approaches, indicating that the additional cost is only paid for at the preprocessing stage. For companies that rely heavily on multi-reasoning queries, this one-time cost per document may be justified, particularly when documents are relatively static and the cost can be amortized over many queries.

The importance of strong multi-hop performance cannot be overstated when it comes to enterprise knowledge bases, since queries often rely on information originating from multiple different sources. The agentic-1 approach highlights that document-level awareness improves cross-reference retrieval, which is crucial for agentic systems. Based on these results, we select agentic-1 as our optimal chunking strategy and use it as the baseline for the experiments conducted in Section 5.2.

5.2 Graph-augmented Retrieval Results

Through the fair comparison conducted in the previous section, we have identified agentic-1 (hierarchical chunking + contextual enhancement) as the optimal strategy on our document corpus. In a next step, we evaluate whether graph-augmented retrieval can further improve performance, particularly on multi-hop queries that require gathering information from multiple different sources. We replicate the GraphRAG framework using AWS infrastructure, constructing knowledge graphs that link entities, chunks, and documents through relationship edges, generated by LLMs. To isolate the effect of the graph structure, we make a head-to-head comparison of three configurations: (1) agentic-1 without graph augmentation, (2) simple hierarchical chunking + graph augmentation, and (3) hierarchical chunking + agentic context enrichment + graph augmentation.

When considering the average performance of the three setups across all datasets, we can see that graph augmentation does not uniformly improve retrieval performance.

Looking at Figure 4, we can see that across all four key metrics, the plain agentic approach discussed in Section 5.1 outperforms both graph approaches. The most significant drop in performance can be observed for the T1EM metric, where both graph-augmented methods (25.4% and 21.0% respectively) achieve only slightly better than half of what agentic returns (40.8%). Since the four key metrics are mainly oriented towards retrieving the precise correct chunk, we can assume that graph traversal reduces precision at the top of the ranking. In addition, out of both graph methods, the combination of graph and hierarchical chunking consistently outperforms the combination of graph and agentic chunking, suggesting that graph construction and contextual summaries do not complement each other well. A possible reason for this could be a phenomenon known as *summary drift*, which means that LLM-generated context references entities and concepts that did not appear in the original chunk text. Thus, upon graph construction, edges might be created based on entities that are not even in the ground-truth content of a chunk. In practice, these "phantom edges" can cause retrieval to return chunks connected to summary entities rather than the actual content entities relevant to the query.

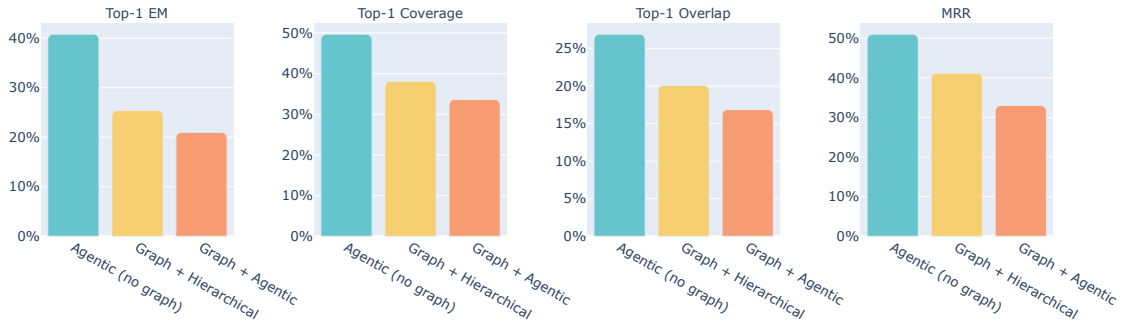

Figure 4: Graph-augmented retrieval performance comparison across key metrics. Metrics are averaged across the three datasets.

When focusing solely on the two multi-hop datasets, we get a better overview of both the strengths and weaknesses of graph-augmented retrieval. Taking all metrics (not only the four key ones) into consideration, we can see a real precision-recall trade-off, since graph augmentation tends to sacrifice top-1 precision in exchange for improved coverage across our corpus. Table 3 shows that, on both datasets, the pure agentic approach achieves a T1EM of 46.9%, substantially outperforming both graph methods (31.8% and 24.8% respectively). On the other hand, when extending the number of considered candidate chunks, graph + hierarchical reports higher scores in metrics that emphasize overall retrieval (not only the top-ranked chunk), such as T5EM, MCC, and BO.

Overall, the findings suggest that graph-augmented retrieval, at least as implemented in the standard GraphRAG framework, does not improve top-ranked retrieval quality. While graphs thrive at expanding the candidate pool to include relevant chunks, they struggle to prioritize them in the correct order. Moreover, they also tend to be 4-5 times slower than non-graph solutions, showcasing a latency of around 1s per query. Based on these results, we will proceed with the agentic approach identified in Section 5.1 as our baseline for the experiments presented in Section 5.3, concluding that the added

	SF2H							MF2H						
Configuration	T1EM	T5EM	T1C	MC	MRR	T1O	BO	T1EM	T5EM	T1C	MC	MRR	T1O	BO
Best Agentic (no graph)	**0.469**	0.747	**0.509**	0.756	**0.581**	**0.315**	0.379	**0.394**	0.670	**0.450**	0.717	**0.505**	**0.279**	**0.293**
Graph + Hierarchical	0.318	**0.752**	0.398	**0.757**	0.483	0.264	**0.390**	0.285	**0.729**	0.367	**0.721**	0.417	0.207	0.273
Graph + Agentic	0.248	0.574	0.333	0.635	0.374	0.208	0.299	0.303	0.563	0.361	0.641	0.386	0.209	0.247

Table 3: Performance breakdown on multi-hop datasets per configuration, considering both graph-augmented and classical setups. Metrics include Top-1 Exact Match (T1EM), Top-5 Exact Match (T5EM), Top-1 Coverage (T1C), Multi-chunk Coverage (MCC), Mean Reciprocal Rank (MRR), Top-1 Overlap (T1O), and Best Overlap (BO). Graph methods improve recall-oriented metrics (T5EM, MCC, BO) but degrade precision-oriented metrics (T1EM, T1C, MRR, T1O).

complexity and computational cost of graph construction and traversal are not justified by the observed performance trade-offs.

5.3 Hybrid Retrieval Results

Having determined the agentic-1 approach as the best-performing retrieval setup so far, we now take things a step further and analyze whether the combination of sparse (BM25) and dense (semantic) retrieval signals can further improve performance. While most existing methods use a fixed weight α for all queries, we have decided that a query-specific approach may bring further improvement. Therefore, we evaluate three approaches of increasing complexity to learn and predict query-level fusion weights: ridge regression (linear model with 4 query-level features), XGBoost (gradient boosting with 25 features including chunk retrieval scores), and MLP (neural network operating on 1024-dimensional query embeddings). It is worth noting that for each of these models, we chose the hyperparameters leading to best predictive accuracy, in terms of R^2. Furthermore, we also evaluate the performance of fixed weight configurations ($\alpha \in \{0, 0.3, 0.5, 0.7, 1\}$, as well as the oracle α^* (the closed-form optimal solution presented in Section 3.3.3.2), to guarantee a fair comparison.

Figure 5 shows the average absolute value of the 4 coefficients considered for ridge regression. One thing that becomes clear immediately is that, on average, the coefficients obtained from the closed-form ridge regression solution are extremely small, all of them being on the scale between 0-0.004. This suggests that the relationship between these query-level features and the optimal fusion weight α^* is either highly non-linear or cannot be explained by such simple features. Thus, this hints towards the fact that ridge regression is unable to learn meaningful patterns from the data, a hypothesis we aim to confirm when evaluating the prediction quality.

Meanwhile, Figure 6 shows the SHAP (SHapley Additive exPlanations) values derived from the trained XGBoost model. This analysis not only reveals which features are the most impactful when it comes to predicting α^*, it also provides the directional importance. Here, it is noteworthy that while both the semantic and BM25 scores had the highest SHAP values, we have decided to only present the four simple features in this plot, to make this analysis as interpretable as possible. The query word count emerges as the most important feature, exhibiting the widest range SHAP values spanning from -0.04 to +0.08. In terms of directional importance, longer queries (red dots) are linked to positive SHAP values, leading to an increase in the predicted α and thus favoring semantic search. Meanwhile, shorter queries (blue dots) produce negative SHAP values,

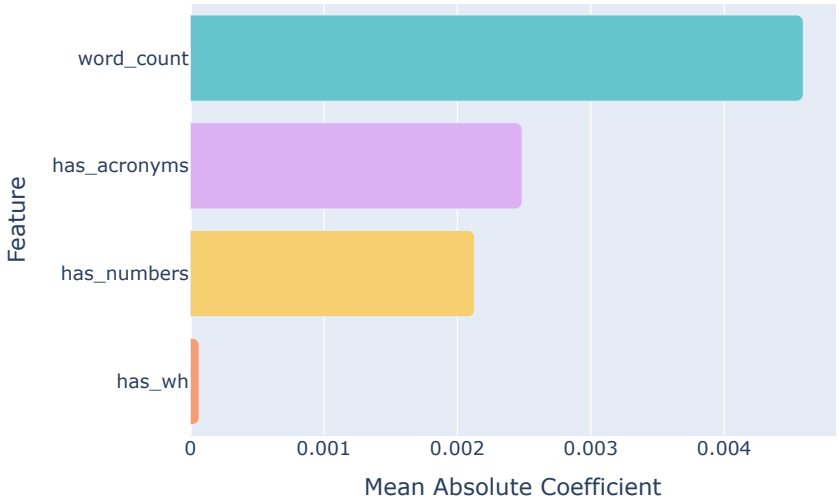

Figure 5: Mean absolute coefficients per feature obtained through the closed-form solution of ridge regression. Best hyperparameter configuration: $\lambda = 100$.

thus favoring BM25. This is more or less aligns with our intuition that longer, descriptive queries benefit from semantic similarity, while while short keyword queries rely on exact term matching. As far as the remaining features are concerned, their SHAP values are closely clustered around 0, suggesting that they only have minimal predictive power. The main difference between XGBoost and the previously analyzed ridge regression approach lies in the fact that XGBoost thrives at capturing complex, highly non-linear relationships.

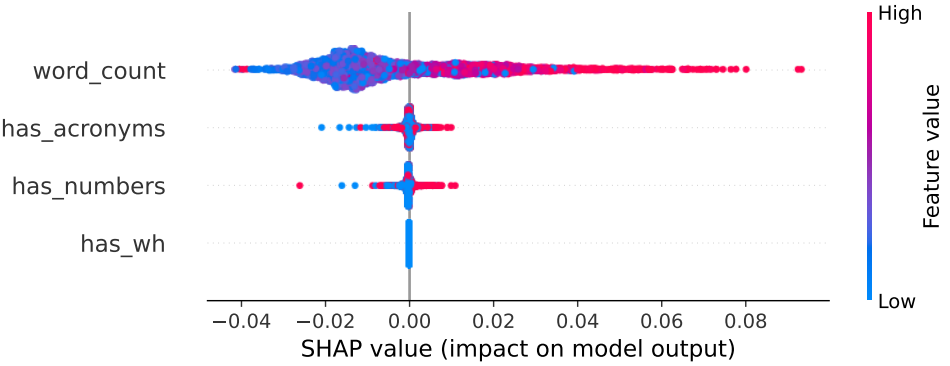

Figure 6: XGBoost SHAP feature importance analysis. Each point represents a query, colored by feature value (blue=low, red=high). The x-axis shows SHAP values indicating impact on predicted α. Best hyperparameter configuration: `n_estimators`=100, `max_depth`=7, `learning_rate`=0.1, `subsample`=0.8.

Figure 7 shows both the training and validation loss curves in the case of the trained MLP. Here, we follow best practices by including early stopping, with the validation loss converging smoothly around epoch 5-10, successfully preventing overfitting. Overall, we see that the MLP struggles to learn any significant patterns, since the final validation

loss remains relatively high, an observation that will be confirmed in the α^* prediction analysis. Two potential reasons for this could either be that the 1˙024-dimensional Titan embeddings do not encode sufficient information when it comes to predicting α^*, or that the model is trained on insufficient data (here we used ∼24˙000 queries during training), posing difficulties to compress high-dimensional vectors to scalar α values.

Figure 7: MLP training and validation loss curves over 30 epochs, using the MSE loss. Best hyperparameter configuration: `hidden_dims`=(128, 64), `learning_rate`=0.001, `batch_size`=128, `dropout`=0.5, `weight_decay`=0.01.

Figure 8 presents the predictive performance of each model by comparing the predicted *alpha* values against the optimal α^* values computed via the closed-form solution. With an extremely low R^2 of 0.002, ridge regression produces predictions that are essentially uncorrelated from α^*, with all predicted α values lying between 0.4-0.5. This confirms our previous assumption, based on the near-zero coefficients, that the relationship between the query-specific features and the fusion weights is more complex than simple linearity. XGBoost achieves higher predictive accuracy, with an R^2 of 0.589 indicating that the model explains approximately 59% of the variance in optimal α^*, highlighting that non-linear modeling combined with retrieval scores can lead to more meaningful predictions. MLP surprisingly underperforms XGBoost despite operating on much richer data (1024-dimensional embeddings vs. 25 query-level features). The low R^2 score of 0.215 suggests that raw embeddings may be powerful when it comes to semantic search, however, they do not encode the necessary information needed to estimate prediction weights as efficiently as simpler, extractable features.

It is important to note that the optimal α^* originates from the logic of using the LSS as a proxy for what the fusion score should reflect (see 3.3.3.2 for more details). In other words, α^* is derived by minimizing the MSE between the fused scores and the LSS, which is only one of many metrics used to measure the retrieval quality of a system.

Table 4 gives a clearer overview of the retrieval quality across the three datasets and four key metrics. For single-hop questions, pure BM25 ($\alpha = 0$) gives the best results,

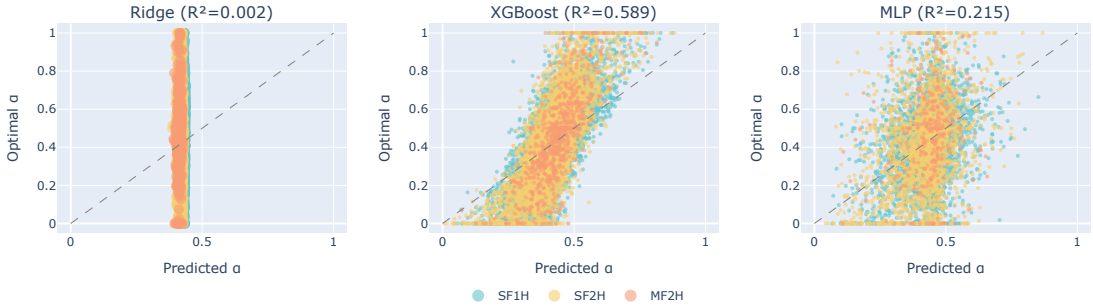

Figure 8: Predicted α vs. optimal α^* scatter plots for ridge regression, XGBoost, and MLP models, broken down by dataset (SF1H, SF2H, MF2H). Diagonal line represents perfect prediction.

achieving a T1EM of 57.3%, 30.9% higher than the 26.4% provided by pure semantic search ($\alpha = 1$). This highlights BM25's strength for keyword-focused, single-hop queries where exact term matching thrives. This gap fades as we move on to multi-hop questions, where we require information from multiple different chunks, with the difference being 17.8% (55.2% vs. 37.4%) on SF2H and 4.2% (40.2% vs. 36.0%) on MF2H. In other words, BM25 might correctly identify one chunk but miss the other if it uses different terminology or does not mention the term at all. On the other hand, semantic search is able to retrieve semantically similar chunks even without keyword overlap, becoming increasingly valuable as queries become more complex.

A further observation is that learned models are competitive or even outperform most fixed α setups on multi-hop datasets. Out of the three models, XGBoost returns the best results, presenting itself as the best-performing predictor across the majority of metrics on multi-hop queries, while showing only a relatively small degradation of 3.6% for T1EM on SF1H with respect to $\alpha = 0$. While for keyword-heavy, single-hop queries BM25 is the optimal strategy to use, the trained XGBoost model provides a more robust general solution that is able to adapt to any query type, behaving similar to BM25 for simple questions, while shifting toward semantic (or balanced) fusion for multi-hop scenarios.

As far as the other two models are concerned, their poor α^* prediction ($R^2 = 0.002$ for ridge regression and $R^2 = 0.215$ for the MLP) does not necessarily reflect as poorly in their retrieval performance. For ridge regression, this is due to the fact that it simply predicts α values close to the mean of the training distribution, estimating a reasonable fixed α without learning query-specific patterns. However, since it offers no clear advantage over simply choosing a fixed α (e.g. $\alpha = 0.3$), it can be considered a failed experiment. For MLP, the performance tends to be slightly better than ridge regression, however, coming at a much higher computational cost and longer training time.

Out of all the configurations, XGBoost is the closest to the outcomes obtained when considering the oracle α^*. Nevertheless, there still seems to be further room for improvement, with the oracle T1EM scores of 56.1% (SF1H), 62.2% (SF2H), and 46.5% (MF2H), 2.4% to 6.6% higher than XGBoost. This gap hints that more expressive features, more training data, or even more sophisticated models could help push the learned fusion closer to the theoretical upper bound. Nevertheless, this upper bound remains "theoretical" as it has access to the ground-truth answers, therefore, the actual ceiling is most likely lower.

For this reason, moving forward, we will use the combination of agentic-1 and learned fusion scores through the trained XGBoost model as our baseline retrieval system.

	SF1H				SF2H				MF2H			
Configuration	T1EM	T1C	T1O	MRR	T1EM	T1C	T1O	MRR	T1EM	T1C	T1O	MRR
$\alpha = 0$	**0.573**	**0.618**	**0.247**	**0.649**	0.552	0.561	0.265	0.671	0.402	0.383	**0.263**	0.503
$\alpha = 0.3$	0.547	0.604	0.242	0.636	0.544	0.555	0.256	0.654	0.433	0.398	0.254	0.527
$\alpha = 0.5$	0.458	0.545	0.213	0.583	0.493	0.511	0.237	0.635	0.435	0.415	0.240	0.540
$\alpha = 0.7$	0.337	0.457	0.171	0.446	0.426	0.456	0.207	0.547	0.366	0.399	0.205	0.473
$\alpha = 1$	0.264	0.397	0.146	0.365	0.374	0.419	0.184	0.487	0.360	0.406	0.180	0.455
Learned Ridge	0.530	0.598	0.235	0.626	0.551	0.560	0.264	0.669	0.427	0.418	0.244	0.531
Learned XGBoost	0.537	0.601	0.239	0.632	**0.556**	**0.562**	**0.266**	**0.672**	**0.439**	**0.421**	0.236	**0.542**
Learned MLP	0.535	0.599	0.237	0.630	0.552	**0.562**	0.265	0.672	0.421	0.413	0.247	0.535
Oracle α^*	0.561	0.639	0.246	0.647	0.622	0.624	0.274	0.717	0.465	0.501	0.255	0.568

Table 4: Hybrid retrieval performance comparing fixed fusion weights (α=0 to 1.0), learned models (Ridge, XGBoost, MLP), and Oracle upper bound (optimal α^* per query), across all three datasets and four key metrics.

5.4 Reranking Results

After having identified XGBoost as an efficient way to estimate the hybrid fusion weight α, we can a take a further post-retrieval step, known as LLM-based reranking. As mentioned before, this approach can add substantial overhead, in terms of cost and latency, since each reranking operation requires a full LLM-invocation. This motivates the need for a conditional reranking strategy, triggered only when the maximum fusion score of a query is below a certain threshold τ.

Figure 9 reveals the uplift in terms of T1EM based on the different reranking thresholds τ. Since most queries have a maximum fusion score of above 0.5, we are most interested in analyzing the outcome for thresholds above 0.5 ($\approx$ never rerank). In the case where we never rerank, the system which leverages XGBoost achieves a T1EM of 53.7%. As we increase the threshold, i.e. as we progressively rerank more queries, reaching around 60.6% at $\tau = 0.8$, before plateauing at 63.2% when considering $\tau = 1$, i.e. always reranking. Overall, this close to 10% accuracy increase demonstrates LLM-based reranking can bring impactful gains.

Meanwhile, the right-hand plot illustrates the incremental uplift with respect to the previous threshold, revealing diminishing gains. The curve seems to show an elbow around the threshold $\tau = 0.8$, where the incremental gain drops significantly to below 1.5% and never manages to break this uplift upper bound for higher thresholds. These decreasing returns pattern suggest that queries with moderately low fusion scores (e.g. < 0.8) benefit the most from reranking, while the retrieved chunks with high fusion scores are more reliable and are the often the correct ones.

In a next step, we look at the precision of skipped queries, i.e. those with sufficiently high fusion scores that we choose not to rerank. If skipped queries show poor accuracy despite having high fusion scores, then it would most likely be wiser to rerank them too. Figure 10 plots the Wilson score intervals, presented in Section 3.4.2.2. For a given threshold, the points lying between the skipped precision, i.e. the proportion of skipped queries that have an exact, ground-truth match in the top-5 ranked chunks, lies in the

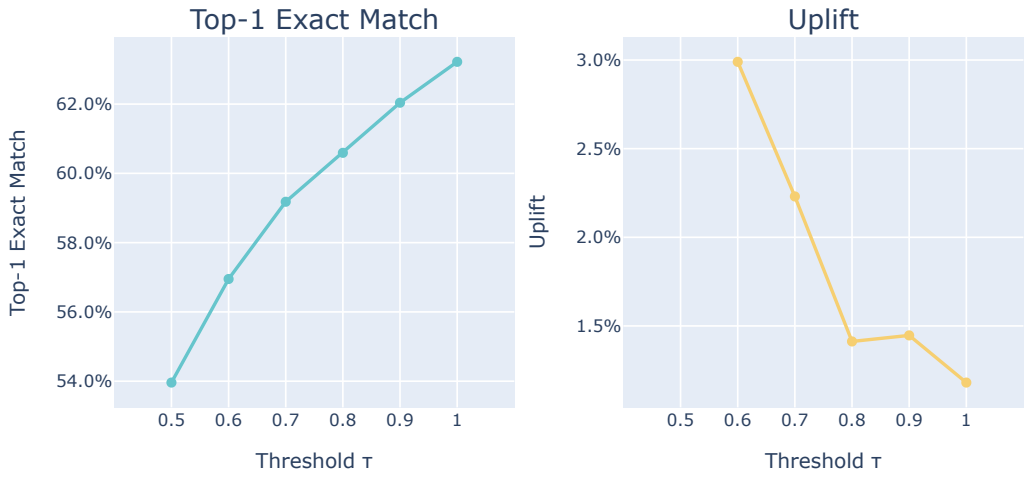

Figure 9: Conditional reranking performance vs. threshold τ. Left plot: Top-1 Exact Match (T1EM) vs. τ. Right plot: T1EM uplift vs. τ.

confidence interval with a confidence level of 95%. When analyzing the figure, we notice that the fusion score is in fact a reliable indicator of the accuracy of a query. At a threshold of $\tau = 0.8$, skipped queries (those with $s_{\max} \geq 0.8$) achieve a T5EM of 39.8%, with a 95% confidence interval lower bound of 38.9% based on a large sample (12˙054 queries out of roughly 34˙000). As the threshold increases further, this skip-level precision increases significantly: 46.5% for $\tau = 0.9$, 54.6% for $\tau = 0.95$, and 62.7% for $\tau = 1$. This progressive increase in precision confirms the previous hypothesis that high fusion scores and better retrieval performance are highly correlated. It is worth noting that the confidence intervals become wider as the thresholds are increased due to smaller sample sizes.

Based on this analysis, we select $\tau^* = 0.95$ as our optimal reranking threshold for two main reasons. First of all, Figure 9 exhibits an elbow right around $\tau = 0.8$, indicating that the returned marginal gains begin to stagnate from that point on. Therefore, we may consider thresholds above or equal to 0.8 as valid candidates. Second, Figure 10 allows us to identify a specific threshold so that we know with a 95% certainty level that the skipped precision will lie above a certain β precision. In our particular case, we see $\beta = 0.5$ as a good precision to strive for, therefore $\tau = 0.95$ is the lowest threshold for which the lower bound of the Wilson score interval is greater than 0.5. In total, this would translate to skipping 5˙655 queries in the combined datasets, representing around 16.6% of all QAs.

From a cost and latency point of view, Table 5 shows that conditional reranking with a threshold of $\tau = 0.95$ would decrease both latency and cost by roughly 20% per query. In the case of an enterprise looking to perform around 1 million queries per month, this would translate to saving \$4˙300 per month. In most scenarios, the query latency plays an even more important role than the accumulated cost, as customer-facing applications are expected to provide meaningful responses within seconds. Therefore, managing to decrease the query latency from 1.20s to 0.96s could improve the overall user experience.

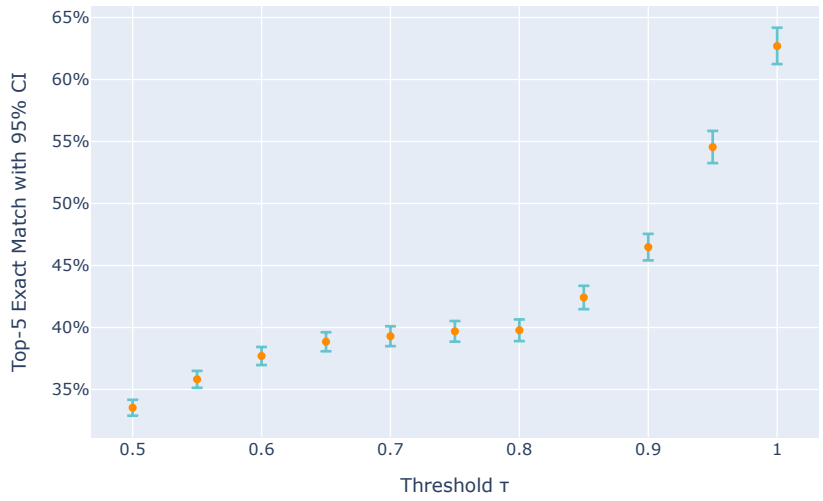

Figure 10: Top-5 Exact Match with 95% Wilson confidence intervals for skipped queries precision.

While lower thresholds show even more promising cost benefits, it is important to keep the previously assessed accuracy in mind when making the trade-off.

Threshold	0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95	1.00
Cost ($/query)	0.0001	0.0016	0.0031	0.0042	0.0051	0.0060	0.0068	0.0080	0.0096	0.0117	0.0160
Reranking latency (s/query)	0.0087	0.1176	0.2297	0.3176	0.3832	0.4474	0.5118	0.5977	0.7220	0.9612	1.2000

Table 5: Cost and reranking latency per query at different confidence thresholds.

5.5 Overall System Performance

Following our thorough analysis of the different retrieval approaches, we now summarize the overall cumulative impact across the entire pipeline. Figure 11 presents a clear overview of the retrieval system performance across four key metrics (Top-1 Exact Match, Top-1 Coverage, MRR, Top-1 Overlap) and three datasets (SF1H, SF2H, MF2H), tracking progression through four pipeline stages: (1) Agentic chunking with semantic search, (2) Graph-augmented retrieval, (3) XGBoost-based hybrid fusion, and (4) Conditional LLM reranking at $\tau^* = 0.95$. By making the right choices in terms of infrastructure, we can see that for Top-1 Exact Match, the system improves from 40.8% (agentic chunking with pure semantic search) to 62.0% (full pipeline with fusion and conditional reranking), translating to a 52% relative improvement. This underlines the need for thoughtful system design across chunking, fusion strategies, and selective reranking.

There are several patterns that emerge consistently across metrics and datasets. First of all, agentic chunking significantly outperforms the standard baseline configurations commonly used in AWS Bedrock implementations, such as basic chunking methods (fixed-window, semantic, hierarchical) combined with pure semantic search. Second, graph-

augmented retrieval methods consistently underperform the agentic baseline across the four key metrics, while showing its strengths when regarding recall-oriented metrics like Top-5 Exact Match and multi-chunk coverage. Third, XGBoost-based hybrid fusion provides improvements over pure semantic search, especially when considering multi-hop queries, which are crucial in the context of an enterprise retrieval system. Finally, conditional reranking provides another significant boost in terms of the four key metrics, without blindly applying LLM-based reranking to every single query. Since these trends remain valid across all four evaluation metrics, we can conclude that these improvements are of actual significance.

Figure 12 extends our analysis by considering the Top-1 Exact Match performance across the 15 most common research fields in our corpus (selected based on paper count from the Amazon Science publication tags). Overall, performance varies slightly across the different domains, however, the global trends remain more or less the same. Nevertheless, this variation highlights the need to adjust pipeline hyperparameters (e.g. the conditional reranking threshold τ) according to the knowledge domain, rather than assuming universal effectiveness.

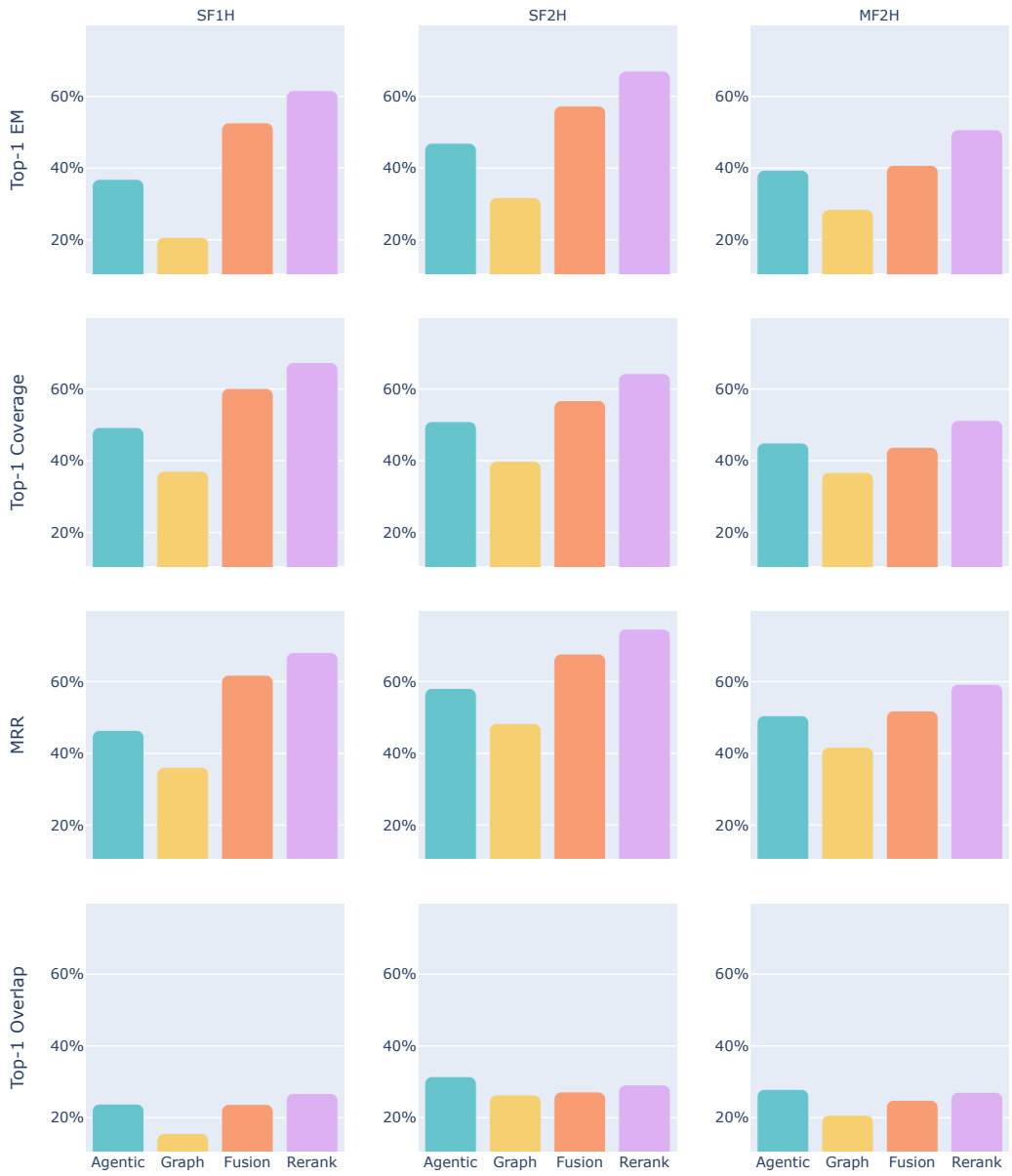

Figure 11: Pipeline progression across key metrics and datasets. Each subplot shows Top-1 Exact Match (T1EM), Top-1 Coverage (T1C), Mean Reciprocal Rank (MRR), or Top-1 Overlap (T1O) for one dataset (SF1H, SF2H, MF2H), with bars representing four pipeline stages: Agentic (contextual chunking + semantic search), Graph (GraphRAG augmentation), Fusion (XGBoost hybrid), and Rerank (conditional LLM reranking with $\tau^* = 0.95$).

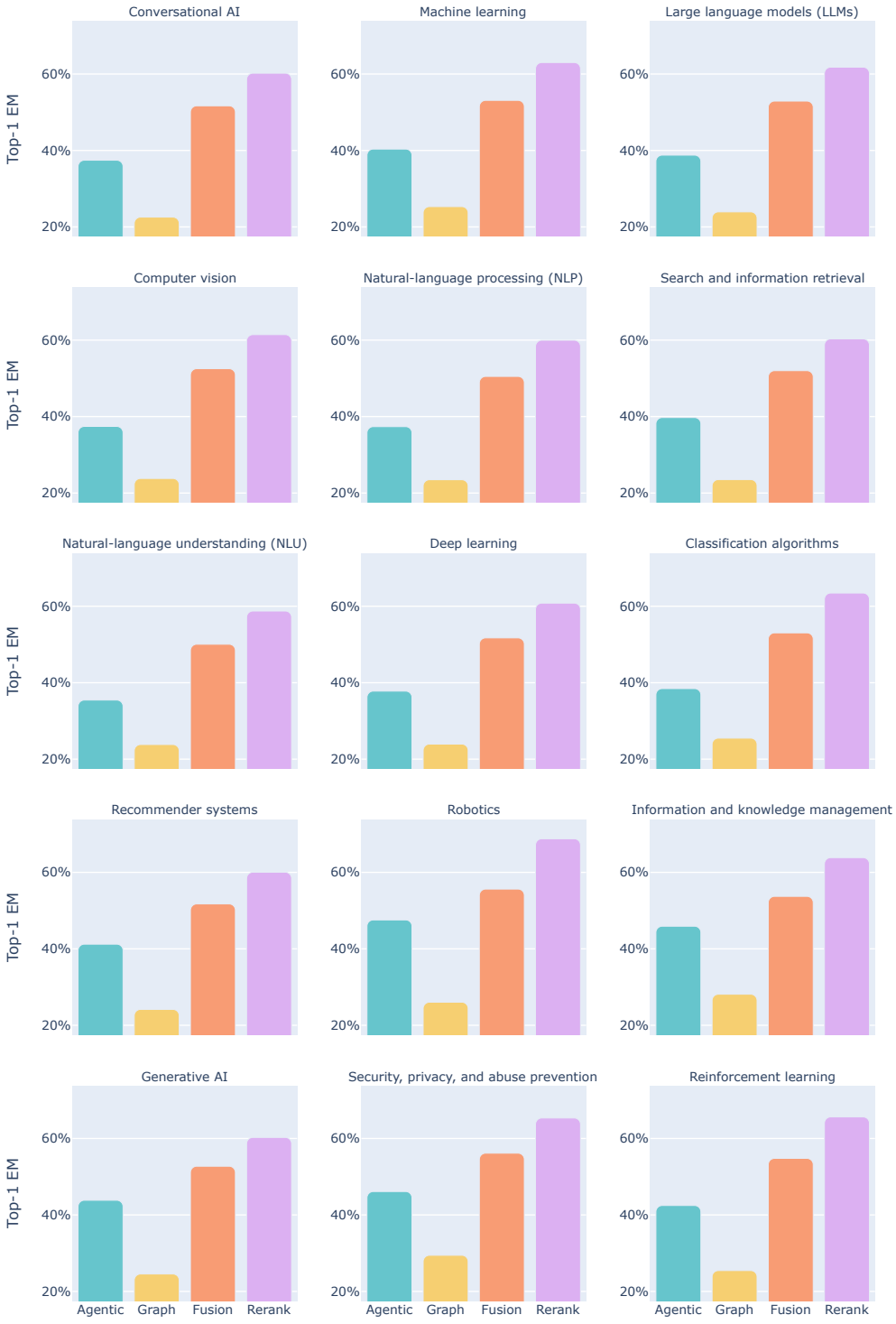

Figure 12: Pipeline progression by research field. Each subplot shows Top-1 Exact Match across the four pipeline stages (Agentic, Graph, Fusion, Rerank) for one of the 15 most common research fields in the corpus (selected by paper count).

6 Conclusion

6.1 Key Findings

This thesis demonstrates that retrieval quality for enterprise knowledge bases, intended to be connected to agentic solutions, heavily depends on the system design throughout the entire pipeline. We introduced four key innovations: (1) agentic chunking, which leverages LLMs to enrich chunks with document-level context, outperforming simple deterministic methods; (2) a systematic evaluation of graph-augmented retrieval showing that GraphRAG degrades top-1 precision in practice, while showing strengths for recall-oriented metrics; (3) query-aware hybrid fusion using XGBoost to dynamically balance BM25 and semantic search based on query characteristics, outperforming fixed-weight approaches on multi-hop queries; (4) conditional reranking with statistically calibrated thresholds, achieving 87% of the accuracy gains of always-rerank while cutting costs and latency by 20%. When considering the full pipeline, Top-1 Exact Match improves from 40.8% to 62.0%, a relative gain of 52%, showing that thoughtful system design across chunking, fusion, and selective reranking can substantially outperform standard configurations. Furthermore, these improvements remain valid across diverse query types (single-hop, multi-hop, cross-document) and research domains, indicating the robustness of these methods. Overall, this work underlines that retrieval infrastructure choices are as important as the underlying LLMs themselves, and organizations deploying agentic systems must invest in end-to-end pipeline optimization rather than relying on default configurations.

6.2 From Research to Production

Within the Supply Chain team at Amazon, we have been working on the implementation of an end-to-end, centralized knowledge base. Since the team has allocated increasing bandwidth to working on agentic solutions, it is essential to have a version-controlled way of curating domain-specific knowledge, creating the digital brain of the team. Therefore, we focused on an architecture consisting of two key components: a user-friendly interface that simplifies the process of uploading and peer-reviewing knowledge files, and a sophisticated backend system that transforms and embeds information, making it readily accessible for agent-based decision-making. As a first step, we opted for the standard configurations across each of the pipeline stages, i.e. basic fixed-window chunking combined with pure semantic search. While this provided us with mediocre results and confirmed that our system was functional, we did not stop there, as we understood the impact that a few minor system design changes could have. We implemented our own customized chunking technique, that leverages markdown structure and semantic boundaries to create meaningful chunks, being particularly well-suited for technical documentation and code repositories. Moreover, we extended this with an adapted version of graph-augmented retrieval, as the majority of our documents are heavily linked to one another and even complement each other. In the future, we aim to continue our research when it comes to additional pipeline stages, such as training a model to predict the optimal balance between sparse and dense signals, as well as identifying a trustworthy conditional reranking threshold.

6.3 Limitations & Future Work

While this work highlights that significant improvements can be made through retrieval system design, there remain several limitations suggesting opportunities for future research. First of all, our analysis relies on self-generated datasets derived from academic research papers rather than established benchmarks. To allow for direct comparison with state-of-the-art retrieval systems, it would be interesting to evaluate our pipeline on public datasets like BEIR [Thakur et al., 2021], MS MARCO [Bajaj et al., 2018], or HotpotQA [Yang et al., 2018]. Second, the steps involving LLM invocations, such as contextual enhancement, segmentation, and reranking, used fairly simple prompts and could benefit from systematic prompt engineering, few-shot examples, or chain-of-thought reasoning to further improve accuracy. Third, our graph-augmented retrieval implementation used the default GraphRAG configurations, leaving the door open for more sophisticated exploration of graph construction strategies. Fourth, the presented pipeline exclusively focuses on processing textual data, without taking images into consideration. This is a critical point, as enterprise documents frequently contain relevant information in figures. Therefore, future work should explore multimodal retrieval using vision-language models for image understanding. Finally, this thesis focuses solely on retrieval performance without measuring end-to-end answer generation quality. In general, it would be interesting to determine whether improved retrieval translates to better agentic outputs and if there is a general methodology to evaluate this impact.

References

[Anthropic, 2024] Anthropic (2024). Introducing contextual retrieval. `https://www.anthropic.com/engineering/contextual-retrieval`. Accessed: 2025-07-28.

[Auer et al., 2024] Auer, C., Lysak, M., Nassar, A., Dolfi, M., Livathinos, N., Vagenas, P., Ramis, C. B., Omenetti, M., Lindlbauer, F., Dinkla, K., Mishra, L., Kim, Y., Gupta, S., de Lima, R. T., Weber, V., Morin, L., Meijer, I., Kuropiatnyk, V., and Staar, P. W. J. (2024). Docling technical report.

[Bajaj et al., 2018] Bajaj, P., Campos, D., Craswell, N., Deng, L., Gao, J., Liu, X., Majumder, R., McNamara, A., Mitra, B., Nguyen, T., Rosenberg, M., Song, X., Stoica, A., Tiwary, S., and Wang, T. (2018). Ms marco: A human generated machine reading comprehension dataset.

[Brown, 2020] Brown, D. (2020). Rank-BM25: A Collection of BM25 Algorithms in Python.

[Bruch et al., 2023] Bruch, S., Gai, S., and Ingber, A. (2023). An analysis of fusion functions for hybrid retrieval. *ACM Transactions on Information Systems*, 42(1):1–35.

[Chen and Guestrin, 2016] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794. ACM.

[Cormack et al., 2009] Cormack, G. V., Clarke, C. L. A., and Büttcher, S. (2009). Reciprocal rank fusion outperforms condorcet and individual rank learning methods. *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*.

[Edge et al., 2025] 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.

[Formal et al., 2021] Formal, T., Piwowarski, B., and Clinchant, S. (2021). Splade: Sparse lexical and expansion model for first stage ranking.

[Gao et al., 2022] Gao, L., Ma, X., Lin, J., and Callan, J. (2022). Precise zero-shot dense retrieval without relevance labels.

[Honnibal et al., 2020] Honnibal, M., Montani, I., Van Landeghem, S., and Boyd, A. (2020). spaCy: Industrial-strength Natural Language Processing in Python.

[Izacard et al., 2022] Izacard, G., Caron, M., Hosseini, L., Riedel, S., Bojanowski, P., Joulin, A., and Grave, E. (2022). Unsupervised dense information retrieval with contrastive learning.

[Karpukhin et al., 2020] Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and tau Yih, W. (2020). Dense passage retrieval for open-domain question answering.

[Khattab and Zaharia, 2020] Khattab, O. and Zaharia, M. (2020). Colbert: Efficient and effective passage search via contextualized late interaction over bert.

[Lewis et al., 2020] Lewis, P., Perez, E., Piktus, A., Karpukhin, V., Goyal, N., Kukla, M., Min, S., Wu, L., Petroni, F., Edunov, S., Chen, D., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 9459–9474.

[Loshchilov and Hutter, 2019] Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization.

[Ni et al., 2021] Ni, J., Qu, C., Lu, J., Dai, Z., Ábrego, G. H., Ma, J., Zhao, V. Y., Luan, Y., Hall, K. B., Chang, M.-W., and Yang, Y. (2021). Large dual encoders are generalizable retrievers.

[Nogueira and Cho, 2020] Nogueira, R. and Cho, K. (2020). Passage re-ranking with bert.

[Nogueira et al., 2020] Nogueira, R., Jiang, Z., and Lin, J. (2020). Document ranking with a pretrained sequence-to-sequence model.

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

[Sun et al., 2024] Sun, W., Yan, L., Ma, X., Wang, S., Ren, P., Chen, Z., Yin, D., and Ren, Z. (2024). Is chatgpt good at search? investigating large language models as re-ranking agents.

[Thakur et al., 2021] Thakur, N., Reimers, N., Rücklé, A., Srivastava, A., and Gurevych, I. (2021). Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models.

[Traag et al., 2019] Traag, V. A., Waltman, L., and van Eck, N. J. (2019). From louvain to leiden: guaranteeing well-connected communities. *Scientific Reports*, 9(1).

[Wilson, 1927] Wilson, E. B. (1927). Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association*, 22(158):209–212.

[Wu et al., 2023] Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A. H., White, R. W., Burger, D., and Wang, C. (2023). Autogen: Enabling next-gen llm applications via multi-agent conversation.

[Xiong et al., 2020] Xiong, L., Xiong, C., Li, Y., Tang, K.-F., Liu, J., Bennett, P., Ahmed, J., and Overwijk, A. (2020). Approximate nearest neighbor negative contrastive learning for dense text retrieval.

[Yang et al., 2018] 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.

[Yao et al., 2023] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). React: Synergizing reasoning and acting in language models.

A Data & Code

The generated datasets (SF1H, SF2H, MF2H), used for the experiments, can be found by following the link down below:
`https://drive.google.com/drive/folders/1tj1Q3ERD9Q2w_TMazccAfskxnWScB2eu?usp=drive_link`

The Python code used for the dataset generation, as well as for performing the different experiments and evaluation, is fully open-source and can be found on my GitHub page, linked down below:
`https://github.com/olijacklu/Amazon-Knowledge-Base-Retrieval`