RIT-RAG: NAVIGATING DOCUMENT CORPORA WITH RETRIEVAL-INDUCED TREES

Anonymous authors
Paper under double-blind review

ABSTRACT

Retrieval-augmented generation (RAG) grounds language models in external corpora. Agentic RAG enables iterative search, yet exposes the model to isolated chunks without document structure, making it difficult to distinguish relevant evidence from chunks that merely resemble the query. Structure-aware methods such as PageIndex navigates using the structure but cannot scale to structures of large corpora that cannot fit in the LLM context. Hence they first commit to a single document using a document retriever and cannot recover from a wrong choice. We propose RIT-RAG (Retrieval-Induced Tree RAG), which combines content retrieval with structural navigation. Offline, RIT-RAG builds a tree for each document from its table of contents or sitemap. At query time, it retrieves a broad set of chunks and uses their positions to induce manageable sub-trees, potentially across multiple documents. An LLM agent navigates these sub-trees, selectively reads promising nodes, and reformulates queries when needed. Thus, retrieval proposes where to look, while the agent decides what to read. Across financial, scientific, and customer-support benchmarks, RIT-RAG achieves the highest answer accuracy among vanilla, graph-based, and agentic baselines. On EntDocs Benchmark, our new benchmark of 2.84 million technical-documentation webpages, it improves accuracy by 6.8–11.4 points over the strongest baseline across three LLMs.

1 INTRODUCTION

Retrieval-Augmented Generation (RAG) has emerged as a standard approach for grounding large language models (LLMs) in external knowledge (Lewis et al., 2020; Gao et al., 2024). By retrieving relevant evidence from a corpus and providing it to an LLM, RAG enables models to answer questions using information beyond their parameters. It also supports citation and attribution. As applications increasingly require answering complex questions over large document collections, effective retrieval has become the central challenge.

Recent work has addressed this challenge through *agentic RAG*, in which an LLM iteratively invokes search tools, examines the returned content, and reinvokes the search tools when necessary (Yao et al., 2023; Asai et al., 2024; Jeong et al., 2024; Hui et al., 2026). This enables the model to recover from unsuccessful retrieval attempts and perform multi-hop information seeking. However, agentic RAG operates primarily at the chunk level. Documents are split into independently searchable chunks, and the LLM reasons over the content of the chunks returned by the search tools. It has no explicit representation of how these chunks are organized within the document. Sections, subsections, and their relationships are therefore invisible to the retrieval process. As a result, without structural information, the agent struggles to distinguish the chunks that contain the answer from those that are merely similar in terminology or topic. The resulting context contains more plausible evidence, but not necessarily more useful evidence, which leads to context overload from unfocused retrieval.

A complementary line of work has explored *structure-aware RAG*, exemplified by PageIndex (Vectify AI, 2025). Rather than searching for content directly, these approaches first reason over a document's hierarchical structure, including sections, subsections, and their summaries, and then retrieve content from promising regions. This constitutes a top-down paradigm, proceeding from structure to content. However, the structure of an entire corpus cannot always fit in the context. The system

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

must first select a document and then navigate its hierarchy. This creates an early commitment problem. Before seeing the document's content, the system has limited ability to compare competing documents. A wrong document selection can therefore lead to a coherent but irrelevant navigation path, with no easy way to recover.

These observations reveal a tension between retrieval recall and precision. Agentic RAG casts a wide net, achieving high recall: relevant evidence is often present somewhere among the retrieved chunks. However, precision is low, as the relevant signal is buried among loosely related content, making it difficult for the LLM to identify and use the right evidence. Structure-aware navigation presents the opposite trade-off. By following a narrow structural path, it can achieve high precision when the correct path is chosen. However, recall is limited: an incorrect choice at the document level can prevent the relevant evidence from ever being considered.

We propose **RIT-RAG** (Retrieval-Induced Tree RAG), a framework that addresses the precision–recall trade-off between agentic RAG and structure-aware RAG. RIT-RAG begins with a conventional search index, enabling scalable retrieval over large document collections. Rather than directly exposing the retrieved chunks to the LLM, RIT-RAG uses their metadata, *e.g.*, the position of each chunk in the hierarchy of its document, to induce sub-trees representing the document structure around the retrieved chunks. The LLM is presented with this compact structural view and selects the region most likely to contain the answer. It can then retrieve the contents of that region. If no candidate region provides a promising path, the LLM reformulates the query. The new query retrieves a different set of chunks and induces new sub-trees. Retrieval and navigation therefore form an iterative loop.

This design enables *selective context assembly*. Rather than dumping all retrieved chunks into the context, as in agentic RAG, or committing to a structural path before seeing relevant content, as in structure-aware RAG, RIT-RAG first uses content retrieval to anchor the search and then induces structure around the retrieved evidence. The resulting sub-trees provide the agent with a map of the relevant documents' neighborhoods, allowing it to selectively expand only the regions it considers promising. In this way, RIT-RAG retains the broad coverage of content-based retrieval while exploiting document structure to focus subsequent retrieval.

The results show that RIT-RAG consistently outperforms existing retrieval approaches across both public benchmarks and a large-scale enterprise setting. On the three publicly available datasets (WixQA, FinanceBench, and QASPER), RIT-RAG achieves the highest accuracy on all of them. It improves accuracy by 2.4–10.5 points over agentic RAG with the same retriever, and by 5.0–18.6 points over PageIndex, demonstrating the benefit of combining content-based retrieval with structure-aware reasoning. The gains hold even when the native structure is too coarse: on FinanceBench, we induce finer tables of contents with an LLM, and RIT-RAG still improves accuracy over the strongest baseline by 4.6–9.3 points. To evaluate scalability in a realistic enterprise setting, we introduce the EntDocs Benchmark, a new large-scale benchmark whose corpus, EntDocs, contains 2.84 million webpages from the product documentation of a large enterprise. On this benchmark, where the structure of EntDocs is too large for PageIndex to navigate, RIT-RAG improves accuracy over the strongest baseline by 6.8–11.4 points, demonstrating that its advantages persist even at substantially larger corpus scales.

Our contributions are as follows.

- We propose RIT-RAG, a framework that combines content-based retrieval with structure-aware navigation. RIT-RAG uses retrieved evidence to induce query-specific sub-trees, possibly from multiple documents, enabling an LLM to reason over document structure before selectively retrieving the contents of promising regions.
- We construct the EntDocs Benchmark, a new benchmark over EntDocs, a corpus of 2.84 million webpages from the product documentation of a large enterprise, to evaluate RAG systems at realistic corpus scale. We will release the benchmark, including its corpus and questions, together with the document trees of all our corpora.
- We evaluate RIT-RAG across multiple models and datasets and show that it consistently improves accuracy over existing RAG systems on three public benchmarks and the EntDocs Benchmark. An analysis of the text that each method reads confirms the precision–recall trade-off: agentic RAG has a high recall but a low precision, PageIndex has the opposite, and RIT-RAG balances the two.

2 Related Work

Retrieval-Augmented Generation Retrieval-Augmented Generation (RAG) grounds language models in external knowledge by retrieving relevant information from a corpus before generation (Lewis et al., 2020). Early RAG systems typically retrieve a fixed set of passages using lexical or dense similarity (Robertson & Zaragoza, 2009; Karpukhin et al., 2020) and provide them directly to the language model. Agentic RAG instead allows the model to iteratively reason about what information is needed and issue additional retrieval operations, with approaches including retrieval interleaved with reasoning (Trivedi et al., 2023), adaptive retrieval (Asai et al., 2024; Jiang et al., 2023), iterative retrieval-generation (Shao et al., 2023) and hybrid retrieval techniques involving multiple tools the agent can use to retrieve content (Hui et al., 2026). In all these settings, the model interacts with the corpus only through retrieval, and it never sees how the retrieved chunks are organized within their documents.

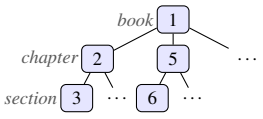
Structured Retrieval A limitation of passage-level retrieval is that independently ranking chunks can discard relationships among pieces of evidence. This has motivated retrieval methods that exploit additional structure in the corpus. One direction constructs **tree-based representations** of the corpus. RAPTOR (Sarthi et al., 2024) recursively clusters and summarizes passages to construct a tree of increasingly abstract representations, while SiReRAG (Zhang et al., 2025) indexes similar and related information to support multihop reasoning. DocsRay (Jeong et al., 2025) generates a pseudo table of contents for a document and uses it for hierarchical retrieval. Another direction constructs **graph-based representations** of the corpus. GraphRAG (Edge et al., 2024) builds an entity graph and community summaries to support retrieval and synthesis over related information, and HippoRAG (Gutiérrez et al., 2024) runs personalized PageRank over a knowledge graph to retrieve passages. HyperGraphRAG (Luo et al., 2025) extends this formulation to hypergraphs, enabling the representation of higher-order relationships among multiple entities or chunks. Cog-RAG (Hu et al., 2026) further constructs complementary theme and entity hypergraphs, using thematic structure for coarse retrieval before fine-grained entity retrieval. A third direction structures the retrieved content at **inference time**. StructRAG (Li et al., 2025) dynamically transforms retrieved information into data structures suited to different reasoning tasks using an LLM. In these approaches, however, the solver model remains oblivious to the underlying structure, and the retriever controls what the model reads rather than the model navigating the structure itself. Moreover, these structures are built by clustering, summarizing, or extracting entities, rather than taken from the organization that the corpus already has.

Structure-Aware Agents A recent line of work instead gives language models access to structure-guided navigation, allowing them to reason about where information is located and decide which parts of the structure to read. PageIndex (Vectify AI, 2025) constructs a hierarchical table of contents for each document and presents the tree of the selected document to the model during inference. The model can then exploit its reasoning capabilities to navigate the tree and read only the sections it deems relevant. PDFTriage (Saad-Falcon et al., 2024) similarly gives the model tools to fetch the sections, pages, and tables of a document. These approaches navigate one document at a time. Over a large corpus, they must first select a document, typically with a retriever, and commit to it before seeing its content.

RIT-RAG avoids this early commitment by combining content-based retrieval with structure-aware navigation. It retrieves from the entire corpus and shows the agent only the sub-trees of the existing hierarchy around the retrieved chunks, which may come from several documents. The agent then decides what to read, which separates evidence discovery from content reading.

3 Preliminaries

Corpus Representation: Let $\mathcal{C} = \{d_1, d_2, \ldots, d_M\}$ denote a corpus containing M documents. Let T_i denote the hierarchical tree structure associated with document d_i. Let V_i be the set of nodes in T_i. These nodes correspond to semantically organised units of the document, such as chapters, sections, and subsections, with parent-child relationships capturing their hierarchical organisation. We represent the corpus by the forest $\{T_1, \ldots, T_M\}$.




```
{ "id": 1,
  "title": "Apple Inc. 2021 Form 10-K",
  "summary": "Annual report for fiscal year 2021.",
  "children": [
    { "id": 2,
      "title": "Part I",
      "children": [
        { "id": 3,
          "title": "Business",
          "keywords": ["iPhone", "Mac", "services"],
          "summary": "Products, segments, and operations."
        },
        ...
       ]},
    { "id": 5,
      "title": "Part II",
      "children": [
        { "id": 6,
          "title": "Management's Discussion and Analysis",
          "keywords": ["MD&A", "results of operations"],
          "summary": "Year-over-year financial review."
        ...
```

Figure 1: An illustrative example of a hierarchical tree structure associated with a document and its JSON string representation.

Each node $v \in V_i$ owns the text that lies under it but not under any of its children, *e.g.*, the introduction of a chapter before its first section. We denote this text by c_v. Note that c_v is empty when all the text of a node lies under its children, *e.g.*, a chapter that begins directly with its first section. Thus, every piece of text in d_i belongs to exactly one node. In addition, each node has associated metadata. The metadata contains a title, a summary of c_v, and keywords present in c_v. The title is mandatory, whereas the summary and keywords are optional and depend primarily on the instructions provided for extracting them and the content available within the node.

Let $\mathcal{S}$ represent the serialization function that converts the structure of the tree T_i into a string $\mathcal{S}(T_i)$. This representation is consumed by the LLM, and it contains just the document's hierarchical structure without exposing the full content of each node. An example is shown in Figure 1. In $\mathcal{S}(T_i)$, each node is a JSON object with a unique identifier, its metadata, and the list of its children in their original order. The LLM uses the identifier to request the content of a node.

The tree structure may be explicitly available or extracted from the documents. For example, a book can be a document whose hierarchy can be obtained by parsing its table of contents and section headings. A group of webpages can be a document whose hierarchical structure can be induced from a sitemap or other organisational metadata.

Agentic Search: Standard vector-index based RAG approaches split each document d_i into manageable chunks, embed each chunk using an off-the-shelf embedding model, such as BGE (Xiao et al., 2024), and store the chunk embeddings in a vector index, *e.g.*, using FAISS (Johnson et al., 2021) or OpenSearch (OpenSearch Project, 2026). During inference, based on its understanding of the user utterance and the ongoing conversation, the agent formulates a search query $\tilde{q}$, and the retriever retrieves the top k chunks, $\mathcal{R}_k(\tilde{q})$, with the highest retrieval scores for $\tilde{q}$. The agent reads the retrieved chunks and decides to either generate the answer a, or formulate another search query to retrieve more chunks. While this iterative process allows the agent to refine its search, the chunk-level index does not preserve the document's hierarchical structure. Consequently, the agent cannot directly leverage structural cues, such as the relationships among sections and subsections, during retrieval. In addition, other than formulating queries, the agent has no control over what it reads, and it reads every retrieved chunk.

PageIndex: Given a user question that requires information from the corpus $\mathcal{C}$, the system first uses a document retriever to identify a document d_i that may contain the answer. The LLM is then provided with the string representation $\mathcal{S}(T_i)$ of the document, which encodes its hierarchical structure without exposing the node contents. The LLM navigates this structure to identify promising nodes and retrieves the content of a node only when needed. It can iteratively inspect additional nodes until it has sufficient evidence to answer the question. Thus, unlike agentic RAG, the LLM has access to the document's structure up front while selectively accessing its contents.

This approach, however, has two limitations. First, since the structures of all documents cannot fit within the LLM's context, the system must first select a document to navigate. This early commitment relies solely on retrieval scores, and the agent cannot switch documents once committed, potentially leading to an incorrect path. Second, the tree representation can become prohibitively large for large (virtual) documents, such as webpage collections. When the full structure exceeds the

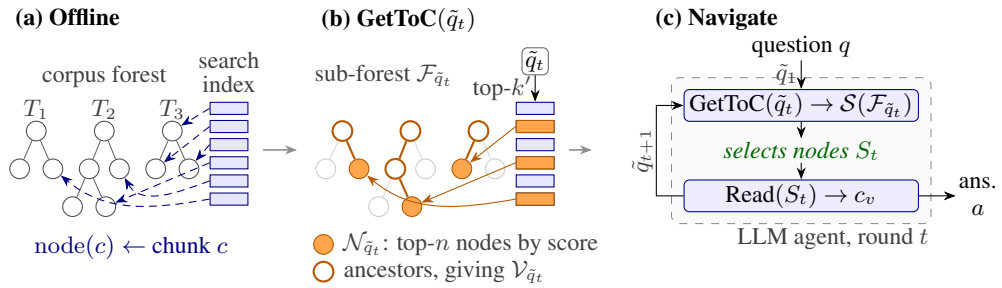


Figure 2: Overview of RIT-RAG. (a) Offline, we index the text of the document trees T_i, with each chunk c belonging to one node $\mathrm{node}(c)$. (b) Each query $\tilde{q}_t$ retrieves top-k' chunks; their top n nodes and ancestors induce a sub-forest $\mathcal{F}_{\tilde{q}_t}$. (c) In each round, the LLM calls GetToC, reasons over $\mathcal{S}(\mathcal{F}_{\tilde{q}_t})$, reads selected nodes S_t, and either answers or continues.

context window, it must be truncated or partitioned, weakening the structural information available to the LLM.

4 RIT-RAG: Retrieval-Induced Tree RAG

Our approach, RIT-RAG, combines content-based retrieval with structure-aware navigation. We first construct a conventional search index over document chunks, as in agentic RAG. In addition to the chunk content, however, we associate each indexed chunk with lightweight structural metadata, including its node identifier, title, and the identifiers and titles of its ancestor nodes in the document tree. This metadata allows the hierarchical structure surrounding a retrieved chunk to be reconstructed at inference time.

Given a user question, the retriever returns the top-$k' > k$ chunks, $\mathcal{R}_{k'}(\tilde{q})$, based on their retrieval scores. Rather than exposing their contents directly to the LLM, we use their structural metadata to reconstruct the corresponding subtrees and provide them as context. The LLM then reasons over these structures to identify promising nodes and selectively request their contents. Thus, structural context is induced by content retrieval, avoiding the need to inspect every document in the corpus.

This design addresses the limitations of both chunk-based agentic RAG and structure-first approaches such as PageIndex. Unlike conventional agentic RAG, the LLM can exploit hierarchical relationships among retrieved nodes when deciding what to retrieve next. At the same time, unlike structure-first approaches, the system does not need to expose or fit the complete structure of the corpus into the LLM context, nor does it need to commit to a single document before considering content-level evidence. Because the initial retrieval operates over the entire corpus, relevant structures from multiple documents can be surfaced simultaneously. The size of the structure presented to the LLM is therefore bounded by the number of selected nodes and their ancestors, rather than by the size of the corpus.

Inducing a Sub-Forest:

We first retrieve the top-$k' > k$ chunks, $\mathcal{R}_{k'}(\tilde{q})$, using the search index. We chunk the text c_v of every node separately, so a chunk never spans multiple nodes. However, a node often spans multiple chunks, especially when it contains a lot of text. Thus, each retrieved chunk c belongs to exactly one node, which we denote by $\mathrm{node}(c)$. We score each node using the highest retrieval score among its retrieved chunks, returned by the index. Formally, the score of a node v is

$$s(v, \tilde{q}) = \max_{\substack{c \in \mathcal{R}_{k'}(\tilde{q}) \\ \mathrm{node}(c)=v}} \mathrm{sim}(\tilde{q}, c). \tag{1}$$

Let $\mathcal{N}_{\tilde{q}}$ be the set of the n nodes with the highest scores, where n is a hyperparameter. To construct a query-dependent sub-forest, we expand $\mathcal{N}_{\tilde{q}}$ with the ancestors of the nodes in it, *i.e.*, all the nodes that lie on the path from any node in $\mathcal{N}_{\tilde{q}}$ to the root of its tree. Let $\mathrm{path}(v)$ denote the set of nodes on the path from v to the root of the tree to which it belongs. Formally,

$$\mathcal{V}_{\tilde{q}} = \bigcup_{v \in \mathcal{N}_{\tilde{q}}} \mathrm{path}(v). \tag{2}$$

The nodes in $\mathcal{V}_{\tilde{q}}$ induce a sub-forest $\mathcal{F}_{\tilde{q}}$, with one sub-tree for every document d_i that contains a node in $\mathcal{N}_{\tilde{q}}$. Each sub-tree is rooted at the root of T_i. We represent $\mathcal{F}_{\tilde{q}}$ as a string $\mathcal{S}(\mathcal{F}_{\tilde{q}})$ by concatenating the string representations of its sub-trees. We list the sub-trees in decreasing order of their score, *i.e.*, the highest score $s(v, \tilde{q})$ among the nodes $v \in \mathcal{N}_{\tilde{q}}$ in the sub-tree. Including the ancestors puts each retrieved node in a global context. For example, two sections titled "Configuring the server" may come from two versions of a product, and only their ancestors distinguish them. In addition, retrieved nodes that share an ancestor appear together under it, which shows the agent where the evidence concentrates.

Navigating the Sub-Forest: In RIT-RAG, the LLM agent works in rounds. In each round t, the agent first formulates a search query $\tilde{q}_t$, and calls the GetToC tool to get $\mathcal{S}(\mathcal{F}_{\tilde{q}_t})$. It then reasons over $\mathcal{S}(\mathcal{F}_{\tilde{q}_t})$ to select a set of nodes S_t, and reads their text using the Read tool. Thus, in each round, we restrict the agent's search to the nodes in $\mathcal{V}_{\tilde{q}_t}$. Based on the contents, c_v for every $v \in S_t$, it may generate the answer a, read more nodes from the same sub-forest $\mathcal{F}_{\tilde{q}_t}$, or formulate another search query for the next round. This is how RIT-RAG recovers from an unpromising sub-forest. Since every GetToC call retrieves from the entire corpus, a reformulated query can surface different documents, and the agent is never committed to the documents of an earlier round. The agent stops when it generates the answer, or when it reaches a budget of B tool calls, of which at most B_{s} can be calls to the GetToC tool, where B and B_{s} are hyperparameters. Figure 2 illustrates this process.

5 Experiments

Our experiments first evaluate how RIT-RAG compares with existing baselines when navigating structure already present in a corpus (RQ1). Such structure, however, is not always available: for example, a PDF may lack a table of contents or provide one that is too coarse to support effective navigation. In such cases, the structure must first be induced. We therefore next evaluate how effectively RIT-RAG navigates an induced structure (RQ2). Finally, we evaluate the scalability of RIT-RAG on a corpus comprising millions of interconnected webpages (RQ3).

5.1 Experimental Setup

Datasets: For RQ1, we use two datasets whose documents come with an existing structure. WixQA (Cohen et al., 2025) has 200 expert-written customer-support questions over 6,221 help-center pages. QASPER (Dasigi et al., 2021) has 1,372 answerable test questions over 416 NLP papers. Since its questions assume a known paper, we prefix each with "According to the paper ⟨title⟩," and index all 1,585 papers from all splits. For RQ2, we use FinanceBench (Islam et al., 2023), with 150 financial questions over 364 SEC filings in PDF. For RQ3, we contribute the EntDocs Benchmark, whose corpus, EntDocs, consists of 2.84 million public webpages crawled from the product documentation of a large enterprise, along with 317 questions on three of its products, generated by Claude Sonnet 5. We will release the EntDocs Benchmark and the document trees of all corpora. Appendix M describes the generation, provenance tracking, and filtering of the EntDocs Benchmark questions. Appendix C reports the statistics of all corpora.

Building the Document Trees: We build the tree of each document from its table of contents (ToC), and the tree of a group of webpages from its sitemap (Section 3, details in Appendix A). In WixQA, the HTML headings of a page give its ToC, and we build a sitemap by grouping the pages into products and the products into categories. In QASPER, the sections of a paper give its ToC. The native ToC of a FinanceBench filing is too coarse, so we induce a finer one with the PageIndex pipeline (Vectify AI, 2025), which parses the PDF and uses an LLM to extract its sections. QASPER and FinanceBench have no sitemap, so their corpora are forests of independent document trees. In EntDocs, the breadcrumbs of the webpages give the sitemap, and each webpage is a single node. Finally, an LLM generates the summary and keywords of every node, except in EntDocs, where the nodes carry only their titles because of the corpus size. To generate summaries, we used Haiku for WixQA, and the open-source Qwen3-32B for QASPER and FinanceBench to reduce cost.

Baselines: Vanilla RAG retrieves the top-5 chunks and answers with a single LLM call. The graph-based methods, HyperGraphRAG (Luo et al., 2025) and Cog-RAG (Hu et al., 2026), extract entities and hyperedges from the corpus offline. For a question, they retrieve the relevant entities, hyperedges, and chunks, and answer from them with a single LLM call. The agentic methods call the

Table 1: Answer accuracy (%) on WixQA, QASPER, and FinanceBench, by generation model. Best per column in **bold**.

Corpus		WixQA			QASPER			FinanceBench		
Family	Method	Haiku	GPT-5-mini	Luna	Haiku	GPT-5-mini	Luna	Haiku	GPT-5-mini	Luna
Vanilla	RAG (k=5)	79.5	86.0	82.5	72.0	77.2	73.7	41.3	49.3	48.0
Graph	HyperGraphRAG	87.5	94.0	87.5	44.8	52.6	52.6	29.3	35.3	47.3
	Cog-RAG	87.0	95.5	92.0	44.4	55.5	52.6	32.0	43.3	45.3
Agentic	Agentic Search	84.5	91.0	91.0	86.2	85.6	85.7	74.0	76.0	75.3
	Interact-RAG	71.5	79.5	85.0	74.2	73.4	78.3	74.7	74.0	80.7
	PageIndex	88.5	86.0	90.5	76.0	75.0	77.0	72.0	75.3	66.7
Ours	RIT-RAG	**95.0**	**96.0**	**95.5**	**88.6**	**88.5**	**89.2**	**84.0**	**84.0**	**85.3**

retriever as a tool in a loop. Agentic Search is the agentic RAG baseline of Section 3, and Interact-RAG (Hui et al., 2026) adds finer control over retrieval, *e.g.*, entity matching. PageIndex (Section 3) also uses the structure: its agent reasons over the full ToC of one document and reads the pages it selects. To select the document, it relies on a document retriever. See Appendix B.2 for the implementation details of document retriever. The graph-based methods and PageIndex preprocess the corpus with the same LLM as RIT-RAG.

Models and Hyperparameters: Every method generates the answer with three LLMs: Claude Haiku 4.5, GPT-5-mini, and GPT-5.6-Luna (Appendix D).. All methods share a hybrid retriever that fuses BM25 (Robertson & Zaragoza, 2009) and dense retrieval with `gte-modernbert-base`(Zhang et al., 2024) and FAISS (Johnson et al., 2021), over chunks of up to 512 tokens (Appendix B.1). The GetToC tool of RIT-RAG retrieves the top-$k' = 200$ chunks and induces the sub-forest from the $n = 50$ highest-scoring nodes (Eq. 1). We choose n on held-out questions (Appendix K). The agent makes at most $B = 10$ tool calls, of which at most $B_{\text{s}} = 3$ are GetToC calls.

Evaluation: Our main metric is accuracy, judged by an LLM (Gemini 2.5 Flash) that compares the prediction with the gold answer. We also measure the precision and recall of the passages that a method reads against the gold evidence, and its preprocessing cost, API cost, and per-query latency. In the appendix, we report lexical token recall (Adlakha et al., 2024) and the seven-dimensional generation evaluation (G-E) of HyperGraphRAG (Luo et al., 2025).

Prompts: Prompts for all LLM based tasks, including different system messages for various baselines, LLM-as-Judge used for evaluation are available in Appendix L

5.2 Results

Performance on Datasets with Existing Structure (RQ1): Table 1 reports the answer accuracy of all methods on WixQA, QASPER, and FinanceBench. RIT-RAG has the highest accuracy on all three datasets, with all three LLMs. WixQA and QASPER come with an existing structure, and they let us test whether navigating this structure helps. The most direct test is to compare RIT-RAG with Agentic Search. Both use the same retriever, the same LLM, and the same budget of $B = 10$ tool calls. The main difference is what the agent sees after a search: Agentic Search sees the retrieved chunks, whereas RIT-RAG sees the sub-forest induced by them. This difference alone improves accuracy by 10.5, 5.0, and 4.5 points on WixQA, and by 2.4, 2.9, and 3.5 points on QASPER, with Haiku, GPT-5-mini, and Luna, respectively. PageIndex also uses the structure, but trails RIT-RAG by 5.0–10.0 points on WixQA and by 12.2–13.5 points on QASPER. It navigates the ToC of a single document, which we select by aggregating the retrieval scores of the chunks of each document (Appendix B.2). Once it selects a document, it cannot fetch the ToC of another one, and if the document is wrong, it cannot reach the evidence (Figure 6). This is the early-commitment problem that we discuss in the introduction. RIT-RAG does not commit to a document, since each search induces a new sub-forest that may span many documents. The graph-based methods are competitive on WixQA, *e.g.*, Cog-RAG reaches 95.5% with GPT-5-mini, but they fall far behind on QASPER, where they are worse than even vanilla RAG (Appendix I). Appendix F reports lexical token recall and the G-E scores, which show similar trends.

Table 4: Precision (P), recall (R), and F1 (%) of the passages that each method reads, measured against the gold evidence, with Haiku as the LLM. Best P, R, and F1 per dataset in **bold**.

Dataset		WixQA			QASPER			FinanceBench		
Family	Method	P	R	F1	P	R	F1	P	R	F1
Vanilla	RAG (k=5)	20.5	64.7	31.1	21.6	67.7	32.8	8.0	34.7	13.0
Agentic	Agentic Search	15.1	70.8	24.9	17.3	**85.5**	28.8	11.5	71.0	19.8
	Interact-RAG	9.1	60.9	15.8	13.8	63.9	22.7	8.8	66.6	15.5
	PageIndex	28.7	72.0	41.0	**54.9**	50.4	52.6	33.1	56.8	41.8
Ours	RIT-RAG	**40.1**	**78.5**	**53.1**	47.9	80.6	**60.1**	**44.7**	**73.2**	**55.5**

Performance on Datasets with Induced Structure (RQ2): On FinanceBench, RIT-RAG navigates the ToCs induced by Qwen3-32B, and still improves over the best baseline by 9.3, 8.0, and 4.6 points. PageIndex navigates the same induced ToCs, but again trails RIT-RAG, by 8.7–18.6 points, as it commits to one retrieved document.

Quality of the Induced Structure: When we induce the structure, the accuracy of a structure-aware method may depend on the performance of the LLM that induces the structure. To test this, we repeat the FinanceBench experiment with Haiku, instead of Qwen3-32B, as the LLM that induces the ToCs and generates the metadata (Table 2). The accuracy of RIT-RAG changes by at most 1.4 points. Thus, RIT-RAG navigates well even the structure induced by a small open-source LLM. Note that the baselines change by up to 4.7 points, although only PageIndex uses the induced ToCs. This is because a chunk never spans two nodes (Section 4), so the induced ToCs also decide the chunks that all methods retrieve. Thus, these baselines are sensitive to how the corpus is chunked, whereas RIT-RAG is not.

Table 2: Accuracy (%) on FinanceBench when Haiku, instead of Qwen3-32B, induces the ToCs. In brackets: the change w.r.t. Qwen3-32B (Table 1).

Family	Method	Haiku	GPT-5	Luna
Vanilla	RAG (k=5)	44.7 (+3.4)	49.3 (0.0)	46.0 (−2.0)
Agentic	Agentic Search	76.7 (+2.7)	80.7 (+4.7)	78.7 (+3.4)
	Interact-RAG	76.0 (+1.3)	76.0 (+2.0)	80.0 (−0.7)
	PageIndex	76.0 (+4.0)	77.3 (+2.0)	66.0 (−0.7)
Ours	RIT-RAG	**84.7** (+0.7)	**84.0** (0.0)	**86.7** (+1.4)

Scaling to Millions of Documents (RQ3): Table 3 reports the accuracy on EntDocs. Here, we compare only with the methods that scale to 2.84 million pages, *viz.*, vanilla RAG, Agentic Search, and Interact-RAG. The graph-based methods need LLM calls over the entire corpus to build their graphs, which is prohibitive at this scale (Section 5.4). PageIndex cannot run either. In FinanceBench, the corpus is a forest of small document trees, so PageIndex can select one tree that fits in the context. In contrast, EntDocs is a single large tree, whose string representation does not fit in the context of any LLM. Moreover, the nodes of EntDocs carry only their titles (Section 5.1). Even so, RIT-RAG improves over the best baseline by 6.6, 7.9, and 11.4 points.

Table 3: Answer accuracy (%) on EntDocs, by generation model. Best per column in **bold**.

Family	Method	Haiku	GPT-5	Luna
Vanilla	RAG (k=5)	43.2	57.4	52.1
Agentic	Agentic Search	73.5	75.4	77.0
	Interact-RAG	74.8	76.0	77.6
Ours	RIT-RAG	**81.4**	**83.9**	**89.0**

5.3 Retrieval Analysis

To understand where the gains of RIT-RAG come from, we look at what each method reads before it answers. Table 4 reports, with Haiku as the LLM, the precision and recall of the passages that each method reads, measured against the gold evidence, and their F1 score. We use WixQA, QASPER, and FinanceBench, where we could run PageIndex. We exclude the graph-based methods, since they also read the descriptions of entities and hyperedges that they create during preprocessing, on which we cannot measure precision and recall (Appendix G).

Agentic Search calls the retriever several times, and reads every chunk that it returns. Thus, it has a high recall, but a low precision. PageIndex shows the opposite trend. It reads only the pages that it selects from a single document, and thus has a high precision. However, if it selects a wrong

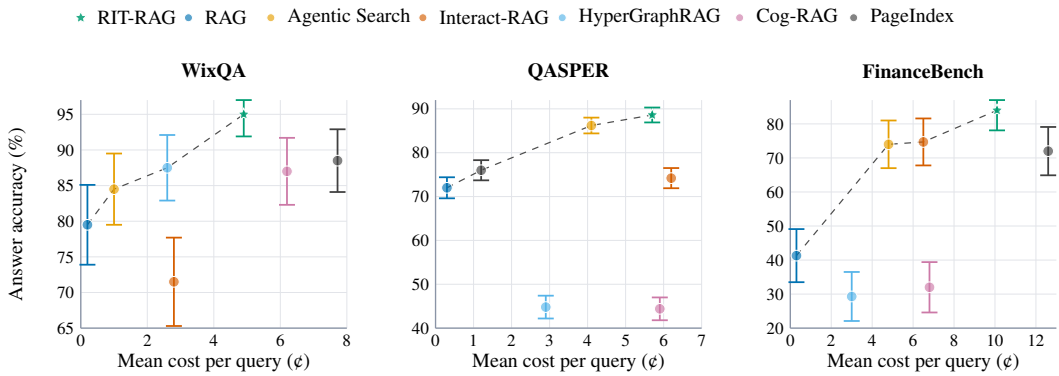


Figure 3: Answer accuracy vs. mean API cost (US cents/question) with Haiku. Each dataset has its own panel/axes. Graph-based methods, PageIndex, and RIT-RAG use the Section 5.1 preprocessing. RIT-RAG is the star; error bars show Wilson 95% CIs, and dashed lines indicate the Pareto frontier.

document, it cannot recover, and its recall drops, *e.g.*, to 50.4% on QASPER. RIT-RAG balances the two. It has the highest precision and recall on WixQA and FinanceBench, and it is second on both on QASPER. As a result, it has the highest F1 on all three datasets. This supports our motivation: the retriever lets the agent search broadly, and the induced sub-forest lets it read selectively. The overlap of correctly answered questions provides further evidence of this complementarity: RIT-RAG retains most of the questions solved exclusively by either PageIndex or Agentic Search (Appendix H). We observe similar trends with GPT-5-mini and Luna (Appendix G).

5.4 Cost

Online Cost: Figure 3 plots the accuracy against the mean API cost per question, with Haiku as the LLM. RIT-RAG is the most accurate method on all three datasets, and it lies on the Pareto frontier. It costs 4.9–10.1 cents per question, more than Agentic Search (1.0–4.8 cents), since the agent reads the string of the induced sub-forest after every search. In return, it improves accuracy by 2.4–10.5 points over Agentic Search. Appendix E compares the latency and the cost of the other baselines.

Offline Cost: Appendix E reports the one-time cost of preprocessing each corpus, at the API prices of Haiku (Table 7). RIT-RAG and PageIndex share the same preprocessing, *viz.*, inducing the ToCs when needed, and generating the node metadata. It costs $15.5 on WixQA and $12.9 on QASPER. On FinanceBench, inducing the ToCs of 364 long filings raises it to $590.1. On EntDocs, it costs nothing, since we use the existing structure and only the titles of the nodes. In contrast, the graph-based methods run LLM extraction over the entire corpus. This costs $85.9–106.6 on WixQA, and grows linearly with the size of the corpus, to an estimated $1.4K–$1.7K on FinanceBench and $30.7K–$38.2K on EntDocs. This is why we do not run them on EntDocs.

6 Conclusion

We presented RIT-RAG, which combines content-based retrieval with structure-aware navigation. Rather than showing the agent the retrieved chunks, or committing to a single document before reading any content, RIT-RAG uses the retrieved chunks to induce query-specific sub-trees from the existing structure of the corpus, and lets the agent decide what to read. On financial, scientific, and customer-support benchmarks, it achieves the highest accuracy against vanilla, graph-based, and agentic baselines, and balances the precision and recall of the text it reads. On the EntDocs Benchmark, with 2.84 million webpages, it improves accuracy by 6.8–11.4 points over the strongest baseline, even though its nodes carry only their titles and it needs no LLM preprocessing. RIT-RAG costs more per question than Agentic Search, since the agent reads the string of the induced sub-trees after every search. Future work could reduce this cost by compacting the string representation, and study richer node metadata at the scale of millions of pages.

Reproducibility Statement

We include the information needed to independently reproduce our experiments in the paper and appendix. The RIT-RAG algorithm and experimental protocol are described in Sections 4 and 5.1, respectively. Details of corpus processing, document-tree construction, and the EntDocs Benchmark dataset are provided in Appendices A, C, and M. Our implementation choices, including node-aligned chunking, hybrid retrieval, reciprocal-rank fusion, and the corpus-level PageIndex adaptation, are documented in Appendices B.1 and B.2. We also report the model and decoding settings, hyperparameter selection procedure, and cost assumptions in Appendices D, K, and E. The prompts used by the agents and evaluators are included in Appendix L, and the answer-quality and retrieval metrics are defined in Appendices F and G.

References

Vaibhav Adlakha, Parishad BehnamGhader, Xing Han Lu, Nicholas Meade, and Siva Reddy. Evaluating correctness and faithfulness of instruction-following models for question answering. *Transactions of the Association for Computational Linguistics*, 12, 2024.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.11511.

Dvir Cohen, Lin Burg, Sviatoslav Pykhnivskyi, Hagit Gur, Stanislav Kovynov, Olga Atzmon, and Gilad Barkan. WixQA: A multi-dataset benchmark for enterprise retrieval-augmented generation. *arXiv preprint arXiv:2505.08643*, 2025.

Gordon V. Cormack, Charles L. A. Clarke, and Stefan Büttcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pp. 758–759, 2009. doi: 10.1145/1571941.1572114.

Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A. Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2021)*, 2021. arXiv:2105.03011.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2024.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024. arXiv:2405.14831.

Hao Hu, Yifan Feng, Ruoxue Li, Rundong Xue, Xingliang Hou, Zhiqiang Tian, Yue Gao, and Shaoyi Du. Cog-RAG: Cognitive-inspired dual-hypergraph with theme alignment retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI 2026)*, 2026. arXiv:2511.13201.

Yulong Hui, Chao Chen, Zhihang Fu, Yihao Liu, Jieping Ye, and Huanchen Zhang. Interact-RAG: Reason and interact with the corpus, beyond black-box retrieval. *arXiv preprint arXiv:2510.27566*, 2026.

Pranab Islam, Anand Kannappan, Douwe Kiela, Rebecca Qian, Nino Scherrer, and Bertie Vidgen. FinanceBench: A new benchmark for financial question answering. *arXiv preprint arXiv:2311.11944*, 2023.

Hyeon Seong Jeong, Sangwoo Jo, Byeong Hyun Yoon, Yoonseok Heo, Haedong Jeong, and Taehoon Kim. Zero-shot document understanding using pseudo table of contents-guided retrieval-augmented generation. *arXiv preprint arXiv:2507.23217*, 2025.

Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C. Park. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 7036–7050, 2024. arXiv:2403.14403.

Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7969–7992, 2023. arXiv:2305.06983.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2021.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, 2020. arXiv:2004.04906.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 2020. arXiv:2005.11401.

Zhuoqun Li, Xuanang Chen, Haiyang Yu, Hongyu Lin, Yaojie Lu, Qiaoyu Tang, Fei Huang, Xianpei Han, Le Sun, and Yongbin Li. StructRAG: Boosting knowledge intensive reasoning of LLMs via inference-time hybrid information structurization. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025. arXiv:2410.08815.

Haoran Luo, Haihong E, Guanting Chen, Yandan Zheng, Xiaobao Wu, Yikai Guo, Qika Lin, Yu Feng, Zemin Kuang, Meina Song, Yifan Zhu, and Luu Anh Tuan. HyperGraphRAG: Retrieval-augmented generation via hypergraph-structured knowledge representation. In *Advances in Neural Information Processing Systems (NeurIPS 2025)*, 2025. arXiv:2503.21322.

OpenAI. text-embedding-3-small. `https://developers.openai.com/api/docs/models/text-embedding-3-small`, 2024.

OpenSearch Project. OpenSearch: Open-source search and analytics suite. Software, `https://opensearch.org`, 2026. accessed 2026.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009. doi: 10.1561/1500000019.

Jon Saad-Falcon, Joe Barrow, Alexa Siu, Ani Nenkova, Seunghyun Yoon, Ryan A. Rossi, and Franck Dernoncourt. PDFTriage: Question answering over long, structured documents. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP), Industry Track*, pp. 153–169, 2024. arXiv:2309.08872.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. RAPTOR: Recursive abstractive processing for tree-organized retrieval. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*, 2024. arXiv:2401.18059.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 9248–9274, 2023. arXiv:2305.15294.

Xueqiao Sun, Xiao Liu, Bowen Lv, Hanchen Zhang, Bohao Jing, Zehan Qi, Yifan Xu, Yuxiao Dong, and Jie Tang. KARL: Reinforcement learning for LLM agents on multi-turn knowledge-intensive agentic tasks. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 10014–10037, 2023. arXiv:2212.10509.

Vectify AI. PageIndex: Document index for vectorless, reasoning-based RAG. GitHub repository, `https://github.com/VectifyAI/PageIndex`, 2025. accessed 2026.

Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-Pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2024)*, 2024. arXiv:2309.07597.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*, 2023. arXiv:2210.03629.

Nan Zhang, Prafulla Kumar Choubey, Alexander Fabbri, Gabriel Bernadett-Shapiro, Rui Zhang, Prasenjit Mitra, Caiming Xiong, and Chien-Sheng Wu. SiReRAG: Indexing similar and related information for multihop reasoning. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025. arXiv:2412.06206.

Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang, Pengjun Xie, Fei Huang, et al. mgte: Generalized long-context text representation and reranking models for multilingual text retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 1393–1412, 2024.

A Corpus-Specific Structure Extraction

Offline, RIT-RAG extracts the corpus strucutre by combining inter-document hierarchy (how documents relate to one another) with intra-document structure induced inside each document (how a document is subdivided into navigable sections). The pipeline is corpus-agnostic except for the parsing step that extracts nodes and parent links from each corpus's native form. The following describes the extraction pipelines for each corpus. Resulting statistics are in Tables 5–6.

WixQA

Inter-document hierarchy is recovered from URL structure. Each article's title and URL identifies its product (170 distinct products). An LLM groups those products into 16 coarse families, giving a three-level backbone family $\rightarrow$ product $\rightarrow$ article. Intra-document structure is induced by splitting each article into sections and subsections using its HTML heading tags. Over the 6,221 help articles this yields 31,733 nodes; article/section body nodes are readable, while the 16 family and 170 product nodes act as navigational containers.

FinanceBench

We treat the full set of 364 SEC filings (10-Ks, 10-Qs, 8-Ks, and annual reports) as a single corpus. Although the benchmark's 150 questions draw on only 84 of these documents, we index all 364 so the remaining filings act as distractors, testing retrieval under realistic corpus scale rather than an oracle-filtered set. The corpus has no authored inter-document hierarchy and hence we treat each filing as independent. Intra-document structure must therefore be induced for every filing. Each native PDF table of contents is too coarse for node-level navigation (individual entries often span tens of pages), so we process each filing with the PageIndex pipeline [1] to produce a finer-grained section hierarchy.

QASPER

Each paper's section hierarchy is recovered directly from its structured markup. Section and subsection headings become tree nodes, giving a shallow (depth $\leq$ 4) per-paper hierarchy without

[1] `https://github.com/VectifyAI/PageIndex`

Table 5: Corpus and evaluation-set statistics. **Docs** is the number of source documents indexed; **Chunks** is the number of retrieval chunks after node-aligned chunking; **Q** is the number of questions evaluated.

Corpus	Node =	Docs	Chunks	Tokens	Q
WixQA	article section	6,221	32,032	3.09 M	200
FinanceBench	PDF page span	364	129,973	48.5 M	150
EntDocs	doc. webpage	2,839,892	8,993,388	3.81 B	317
QASPER	paper section	1,585	31,014	10.6 M	1372

Table 6: Hierarchical document structure per corpus. Depth is from the document root; node size is a node's own-text token count.

Corpus	Total nodes	Max depth	Node size (tok) mean	med.	p95
WixQA	31,733	5	84	60	219
FinanceBench	79,658	8	602	274	2,368
EntDocs	2,934,928	17	1,020	182	1,960
QASPER	24,350	4	362	262	977

additional induction. There is no inter-document hierarchy, as papers are independent research artifacts. Section body nodes are readable. Heading-only container nodes serve as navigational anchors. Because the native section headings are already clean and informative, we only generate node summaries and keywords but do not rewrite node titles.

EntDocs

The EntDocs corpus is derived from the IBM Documentation portal.[2] Starting from the portal homepage, we crawl the documentation while preserving the existing product navigation trees. The structure is entirely inter-document. Each product is organised as a hierarchy of webpages connected through the sitemap There is no intra-document structure to induce, as each webpage corresponds to a single node and is not further subdivided. We combine the per-product trees under a virtual root to obtain a unified corpus containing 2.93M webpages, with a maximum depth of 17.

B Retrieval Implementation

B.1 RIT-RAG

We chunk each node of the corpus $\mathcal{C}$ into chunks of fixed token size of 512. RIT-RAG uses hybrid retriever for scoring a chunk's similarity to the query. To return the top-k chunks, it retrieves the top k' chunks from BM25 and from the dense index, fuses the two rankings with reciprocal rank fusion (Cormack et al., 2009) with with fusion constant $\kappa = 60$, the standard RRF setting combining both the lexical overlap and semantic signals.

B.2 PageIndex

We implement the corpus-level PageIndex following the methodology described in (Vectify AI, 2025). We first retrieve the top k chunks with vector search. For a filing D with n_D retrieved chunks, we compute $\mathrm{DocScore}(D) = \frac{1}{\sqrt{n_D+1}} \sum_c \mathrm{ChunkScore}(c)$, summing over the retrieved chunks c that belong to D. We select the highest-scoring filing, retrieve its table of contents, and pass that structure to the PageIndex agent for within-document navigation.

C Dataset and Structure Statistics

Table 5 reports corpus and benchmark sizes. Table 6 reports the extracted corpus structure.

[2] `https://www.ibm.com/docs/en/products`

Table 7: Models and decoding settings used in the experiments. **Role** identifies how each model is used in the pipeline. **Temperature** and **Max tokens** report the generation settings. **Input/Output price** gives the effective API cost in USD per one million tokens.

Model	Role	Key decoding settings	$/1M (in / out)
Claude Haiku 4.5	Generation + preprocessing	default sampling (512 output tok. for prep)	0.76 / 3.80
GPT-5-mini	Generation	temperature 1; medium reasoning effort	0.25 / 2.00
GPT-5.6-Luna (Luna)	Generation	temperature 1; non-reasoning	1.00 / 6.00
Qwen3-32B	Preprocessing	non-thinking; T=0.7, top-p=0.8, top-k=20, min-p=0; YaRN 64k	0.08 / 0.28
Gemini 2.5 Flash	Evaluation (binary + G-E judges)	T=0	–

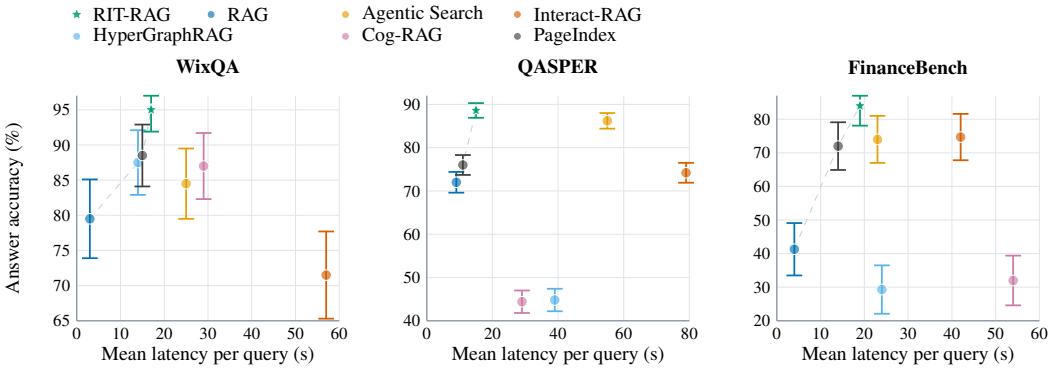


Figure 4: Answer accuracy versus the mean LLM latency per question (seconds), with Haiku as the LLM. Each dataset has its own panel and axes. The graph-based methods, PageIndex, and RIT-RAG use the preprocessing of Section 5.1, *viz.*, Haiku on WixQA, and Qwen3-32B on QASPER and FinanceBench. RIT-RAG is the star, the error bars are Wilson 95% confidence intervals, and the dashed line is the Pareto frontier.

Table 8: Per-query online trade-offs (Haiku generation). Acc.: binary-judge accuracy (%); ¢/q: mean API cost (US cents); s/q: mean LLM latency (s).

	WixQA (n=200)			QASPER (n=1372)			FinanceBench (n=150)		
Method	Acc.	¢/q	s/q	Acc.	¢/q	s/q	Acc.	¢/q	s/q
RAG (k=5)	79.5	0.2	3	72.0	0.3	9	41.3	0.3	4
HyperGraphRAG	87.5	2.6	14	44.8	2.9	39	29.3	3.0	24
Cog-RAG	87.0	6.2	29	44.4	5.9	29	32.0	6.8	54
Agentic Search	84.5	1.0	25	86.2	4.1	55	74.0	4.8	23
Interact-RAG	71.5	2.8	57	74.2	6.2	79	74.7	6.5	42
PageIndex	88.5	7.7	15	76.0	1.2	11	72.0	12.6	14
RIT-RAG	**95.0**	4.9	17	**88.6**	5.7	15	**84.0**	10.1	19

D Model and Inference Configuration

Table 7 lists the language models used for generation and offline preprocessing, together with decoding settings and the per-token prices used for cost accounting in Table 9.

E Cost and Latency Analysis

Table 9 reports the one-time preprocessing cost for each method at the API prices listed in Table 7.

Table 9: One-time preprocessing cost (USD) at the prices of Haiku (Table 7). The other baselines need no preprocessing. †Estimated by scaling linearly from WixQA.

Method	WixQA	QASPER	FinanceBench	EntDocs
HyperGraphRAG	$85.9	∼$294.70†	∼$1.4K†	∼$30.7K†
Cog-RAG	$106.6	∼$365.70†	∼$1.7K†	∼$38.2K†
PageIndex	$15.5	$12.9	$590.1	–
RIT-RAG	$15.5	$12.9	$590.1	$0

Table 10: WixQA answer quality by method and generation model. G-E dimensions (each normalized 0–1): Corr(ectness), Rel(evance), Fact(uality), Compr(ehensiveness), Know(ledgeability), Coher(ence, logical), Div(ersity). Recall is lexical token recall. Best per column in **bold**.

Method	Gen	Recall	Corr	Rel	Fact	Compr	Know	Coher	Div
RAG (k=5)	Haiku	0.427	0.740	0.821	0.779	0.658	0.745	0.948	0.491
	GPT-5-mini	0.411	0.792	0.883	0.796	0.739	0.851	0.929	0.652
	Luna	0.256	0.778	0.850	0.831	0.632	0.742	0.907	0.528
HyperGraphRAG	Haiku	0.632	0.858	0.926	0.860	0.823	0.889	0.945	0.781
	GPT-5-mini	**0.655**	0.969	**0.989**	0.975	0.959	0.978	0.997	0.932
	Luna	0.514	0.919	0.974	0.935	0.882	0.922	0.976	0.792
Cog-RAG	Haiku	0.614	0.849	0.919	0.879	0.829	0.892	0.939	0.834
	GPT-5-mini	0.647	**0.980**	**0.989**	**0.980**	**0.966**	**0.992**	**0.999**	**0.953**
	Luna	0.525	0.946	0.984	0.953	0.907	0.960	0.987	0.831
Agentic Search	Haiku	0.581	0.814	0.926	0.828	0.777	0.862	0.942	0.660
	GPT-5-mini	0.527	0.911	0.952	0.921	0.877	0.929	0.978	0.727
	Luna	0.404	0.922	0.968	0.938	0.840	0.921	0.990	0.679
Interact-RAG	Haiku	0.391	0.731	0.912	0.798	0.766	0.855	0.845	0.726
	GPT-5-mini	0.207	0.849	0.894	0.892	0.861	0.912	0.896	0.906
	Luna	0.352	0.876	0.928	0.914	0.865	0.895	0.954	0.811
PageIndex	Haiku	0.619	0.851	0.917	0.877	0.829	0.884	0.945	0.715
	GPT-5-mini	0.548	0.911	0.945	0.923	0.878	0.917	0.982	0.772
	Luna	0.462	0.925	0.952	0.934	0.867	0.918	0.969	0.714
RIT-RAG	Haiku	0.638	0.872	0.958	0.876	0.869	0.923	0.958	0.650
	GPT-5-mini	0.587	0.948	0.980	0.965	0.930	0.969	0.994	0.677
	Luna	0.474	0.941	0.980	0.956	0.909	0.953	0.986	0.760

At query time, RIT-RAG costs more than vanilla RAG (0.2–0.3 cents per question), but improves accuracy by 15.5– 42.7 points. RIT-RAG is also faster than Agentic Search (15–19 vs. 23–55 seconds) and Interact- RAG (42–79 seconds) (Figure 4). PageIndex is the fastest agentic method (11–15 seconds), and on QASPER, where each document is small, it is also the cheapest (1.2 cents). However, it trails RIT-RAG in accuracy on all three datasets, as it cannot change the document once it commits to it.

F Answer-Quality Metrics

We report lexical token recall and all seven G-E dimensions (Luo et al., 2025) (correctness, relevance, factuality, comprehensiveness, knowledgeability, logical coherence, and diversity; each normalized to 0–1) for every dataset, using one preprocessing model per corpus and all three generation models (Haiku, GPT-5-mini, and Luna; Tables 10–13). All G-E dimensions across every method and generation model are scored by the same (Gemini 2.5 Flash) judge so that the values are comparable within and across rows.

G Retrieval Metrics

We compute F1 as $2PR/(P+R)$, where P and R are the precision and recall of the passages that a method reads, measured against the gold evidence. With Haiku as the LLM (Table 4), Agentic Search has the highest recall on QASPER (85.5%), but a precision of only 11.5–17.3%, lower

Table 11: FinanceBench answer quality by method and generation model. G-E dimensions (each normalized 0–1): Corr(ectness), Rel(evance), Fact(uality), Compr(ehensiveness), Know(ledgeability), Coher(ence, logical), Div(ersity). Recall is lexical token recall. Best per column in **bold**.

Method	Gen	Recall	Corr	Rel	Fact	Compr	Know	Coher	Div
RAG (k=5)	Haiku	0.370	0.791	0.933	0.771	0.881	0.835	0.988	0.433
	GPT-5-mini	0.278	0.705	0.910	0.733	0.843	0.837	0.985	0.455
	Luna	0.255	0.667	0.918	0.677	0.803	0.760	0.993	0.407
HyperGraphRAG	Haiku	0.386	0.515	0.807	0.546	0.682	0.687	0.959	0.599
	GPT-5-mini	0.296	0.574	0.828	0.588	0.645	0.648	0.737	0.407
	Luna	0.338	0.693	0.935	0.683	0.697	0.737	0.778	0.401
Cog-RAG	Haiku	0.417	0.551	0.851	0.582	0.721	0.763	0.985	0.581
	GPT-5-mini	0.336	0.650	0.835	0.632	0.682	0.745	0.809	0.462
	Luna	0.386	0.691	0.899	0.728	0.695	0.754	0.753	0.431
Agentic Search	Haiku	0.545	0.837	0.923	0.856	0.893	0.915	0.982	0.550
	GPT-5-mini	0.425	0.888	0.965	0.885	0.907	0.920	0.992	0.451
	Luna	0.422	0.863	0.941	0.865	0.895	0.914	0.988	0.465
Interact-RAG	Haiku	0.444	0.821	0.927	0.817	0.741	0.831	0.655	0.371
	GPT-5-mini	0.300	0.799	0.915	0.808	0.683	0.799	0.636	0.301
	Luna	0.430	0.881	0.961	0.889	0.851	0.895	0.845	0.405
PageIndex	Haiku	0.548	0.866	0.939	0.865	0.907	0.927	0.976	**0.623**
	GPT-5-mini	0.458	0.885	0.964	0.900	0.926	0.933	0.985	0.492
	Luna	0.300	0.869	0.971	0.873	0.907	0.920	0.992	0.519
RIT-RAG	Haiku	**0.579**	0.913	0.949	0.908	0.935	0.947	0.980	0.620
	GPT-5-mini	0.458	0.923	0.973	0.943	**0.949**	**0.964**	0.995	0.534
	Luna	0.516	**0.941**	**0.977**	**0.951**	0.938	0.963	**0.997**	0.537

Table 12: EntDocs Benchmark answer quality by method and generation model. G-E dimensions (each normalized 0–1): Corr(ectness), Rel(evance), Fact(uality), Compr(ehensiveness), Know(ledgeability), Coher(ence, logical), Div(ersity). Recall is lexical token recall. Best per column in **bold**.

Method	Gen	Recall	Corr	Rel	Fact	Compr	Know	Coher	Div
RAG (k=5)	Haiku	0.392	0.601	0.698	0.626	0.607	0.557	0.928	0.379
	GPT-5-mini	0.288	0.709	0.801	0.721	0.655	0.697	0.880	0.457
	Luna	0.218	0.655	0.734	0.664	0.564	0.608	0.827	0.385
Agentic Search	Haiku	0.559	0.827	0.868	0.832	0.777	0.839	0.918	0.470
	GPT-5-mini	0.369	0.834	0.883	0.851	0.759	0.836	0.929	0.502
	Luna	0.374	0.852	0.891	0.863	0.790	0.845	0.924	0.548
RIT-RAG	Haiku	**0.602**	0.891	0.924	0.890	0.860	0.892	0.933	0.551
	GPT-5-mini	0.447	0.910	0.934	0.914	0.854	0.910	0.968	0.522
	Luna	0.443	**0.937**	**0.966**	**0.938**	**0.898**	**0.942**	**0.971**	**0.637**

than even vanilla RAG on WixQA and QASPER. PageIndex has the highest precision on QASPER (54.9%), but its recall drops to 50.4% on QASPER and 56.8% on FinanceBench. On WixQA, its recall remains 72.0%, possibly because the answer to a question usually lies in a single help page. On QASPER, RIT-RAG is second on both measures, but close to the best: 80.6% vs. 85.5% recall, and 47.9% vs. 54.9% precision. With Luna, RIT-RAG also has the highest F1 on all three datasets. With GPT-5-mini, PageIndex has a slightly higher F1 than RIT-RAG on QASPER (50.0% vs. 47.0%) and FinanceBench (43.3% vs. 42.9%), although RIT-RAG still has the highest recall on all three datasets.

H Complementarity Analysis

We compare the questions answered correctly by RIT-RAG, PageIndex, and Agentic RAG. PageIndex and Agentic RAG have complementary strengths: PageIndex navigates document structure, whereas Agentic RAG searches broadly over content. Figure 5 shows how their correct answers overlap with those of RIT-RAG.

Table 13: QASPER answer quality by method and generation model. G-E dimensions (each normalized 0–1): Corr(ectness), Rel(evance), Fact(uality), Compr(ehensiveness), Know(ledgeability), Coher(ence, logical), Div(ersity). Recall is lexical token recall. Best per column in **bold**.

Method	Gen	Recall	Corr	Rel	Fact	Compr	Know	Coher	Div
RAG (k=5)	Haiku	0.680	0.836	0.925	0.823	0.869	0.822	0.995	0.537
	GPT-5-mini	0.611	0.833	0.913	0.828	0.835	0.846	0.987	0.574
	Luna	0.569	0.789	0.885	0.792	0.779	0.795	0.978	0.516
HyperGraphRAG	Haiku	0.612	0.651	0.765	0.653	0.650	0.694	0.944	0.646
	GPT-5-mini	0.530	0.729	0.841	0.734	0.693	0.760	0.939	0.603
	Luna	0.542	0.752	0.838	0.756	0.707	0.776	0.934	0.605
Cog-RAG	Haiku	0.557	0.588	0.724	0.604	0.600	0.636	0.920	0.606
	GPT-5-mini	0.534	0.745	0.848	0.746	0.709	0.770	0.950	0.615
	Luna	0.525	0.747	0.843	0.755	0.719	0.782	0.949	0.633
Agentic Search	Haiku	0.822	0.897	0.941	0.892	0.907	0.935	0.989	0.685
	GPT-5-mini	0.714	0.897	0.948	0.900	0.900	0.923	**0.997**	0.578
	Luna	0.696	0.896	0.947	0.903	0.890	0.922	0.990	0.657
Interact-RAG	Haiku	0.672	0.798	0.888	0.792	0.764	0.809	0.946	0.560
	GPT-5-mini	0.499	0.773	0.873	0.775	0.748	0.801	0.898	0.568
	Luna	0.555	0.830	0.911	0.829	0.802	0.848	0.951	0.619
PageIndex	Haiku	0.736	0.868	0.902	0.876	0.860	0.880	0.986	0.696
	GPT-5-mini	0.644	0.848	0.927	0.862	0.847	0.859	0.988	0.621
	Luna	0.641	0.862	0.916	0.860	0.848	0.875	0.981	0.685
RIT-RAG	Haiku	**0.836**	0.915	0.945	0.908	0.924	**0.951**	0.990	**0.755**
	GPT-5-mini	0.737	**0.929**	**0.963**	0.929	**0.935**	0.949	**0.997**	0.613
	Luna	0.715	0.923	0.961	**0.930**	0.923	0.947	0.993	0.726

Table 14: Precision (P), recall (R), and F1 (%) of the passages that each method reads, measured against the gold evidence, with GPT-5-mini as the LLM. Best P, R, and F1 per dataset in **bold**.

Dataset		WixQA			QASPER			FinanceBench		
Family	Method	P	R	F1	P	R	F1	P	R	F1
Vanilla	RAG (k=5)	20.5	64.7	31.1	21.6	67.7	32.8	8.0	34.7	13.0
Agentic	Agentic Search	17.1	76.9	28.0	19.0	85.0	31.1	12.9	68.6	21.7
	Interact-RAG	7.2	75.4	13.1	9.6	61.1	16.6	6.9	63.2	12.4
	PageIndex	28.7	72.0	41.0	**50.4**	49.7	**50.0**	**34.5**	58.0	**43.3**
Ours	RIT-RAG	**31.8**	**80.0**	**45.5**	32.3	**86.1**	47.0	29.6	**78.1**	42.9

Table 15: Precision (P), recall (R), and F1 (%) of the passages that each method reads, measured against the gold evidence, with Luna as the LLM. Best P, R, and F1 per dataset in **bold**.

Dataset		WixQA			QASPER			FinanceBench		
Family	Method	P	R	F1	P	R	F1	P	R	F1
Vanilla	RAG (k=5)	20.5	64.7	31.1	21.6	67.7	32.8	8.0	34.7	13.0
Agentic	Agentic Search	16.6	75.4	27.2	19.8	85.6	32.2	13.1	72.8	22.2
	Interact-RAG	12.6	74.8	21.6	13.4	68.1	22.4	10.3	67.3	17.9
	PageIndex	28.7	72.0	41.0	**49.3**	43.9	46.4	**31.6**	50.4	38.8
Ours	RIT-RAG	**32.4**	**80.1**	**46.1**	34.4	**86.4**	**49.2**	30.1	**77.1**	**43.3**

Among the questions answered correctly by PageIndex but not by Agentic RAG, RIT-RAG answers 22 of 27 on WixQA, 11 of 15 on FinanceBench, and 57 of 67 on QASPER (Figure 6). Among those answered correctly by Agentic RAG but not by PageIndex, RIT-RAG answers 9 of 10, 14 of 18, and 168 of 192, respectively (Figure **??**). It also answers 2, 11, and 20 questions that both baselines miss. These results show that RIT-RAG retains most of the distinct strengths of both baselines while also solving some questions that neither baseline answers correctly.

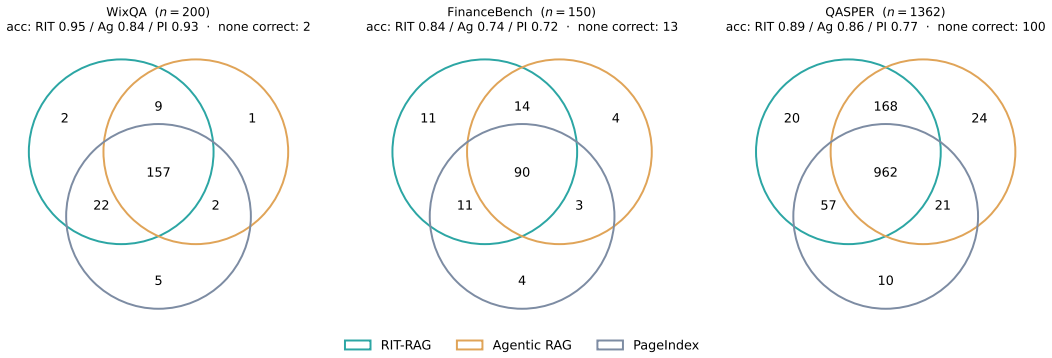


Figure 5: Which questions each method answers correctly (Haiku generation), as an outline Venn. Each colored ring is one method's set of correct questions and region labels are question counts; "none correct" (all three wrong) is noted per panel.

Table 16: Per-query tool-call statistics for RIT-RAG (Haiku generation). **ToC/Cont.**: mean calls to `get_toc_structure`/`get_content`; **Total**: their sum; **Nodes**: mean content nodes per query; **N/call**: nodes per `get_content` call.

		Mean calls per query				
Benchmark	n	Total	ToC	Cont.	Nodes	N/call
WixQA	200	2.99	1.33	1.66	5.5	3.3
QASPER	1372	2.75	1.43	1.32	3.3	2.5
FinanceBench	150	2.70	1.46	1.24	2.7	2.2
EntDocs Benchmark	317	2.99	1.47	1.51	3.8	2.5

I WHY GRAPH BASELINES FAIL ON QASPER AND FINANCEBENCH

The graph baselines are strong on WixQA yet collapse on QASPER and FinanceBench, for two different reasons. Figure 8 gives representative traces from both datasets.

On QASPER the cause is a loss of document scoping. QASPER is per-paper question answering, but HyperGraphRAG builds a single graph over all 1,585 papers with entities keyed by surface form, so the generic vocabulary of scientific writing fuses into corpus-spanning hubs (one `DATASET` node linked to 1,728 passages, `MODEL` 4,046, `BERT` 1,397). A question about one paper then retrieves fragments pooled from unrelated papers, and the model copies the wrong paper's facts (Figure 8*(a)*).

On FinanceBench the answers live in tables, but graph extraction shreds each table into fragments and stores bare figures as entities (about a fifth of all nodes are pure numbers, which are poor retrieval keys), so the gold evidence is rarely surfaced, resulting in hallucinations (Figure 8*(b)*).

J RIT-RAG BEHAVIOR STATISTICS

Table 16 reports per-query tool-call statistics for RIT-RAG on all four benchmarks under Haiku generation. RIT-RAG navigates with two tools, GetToC and Read.

Tool calls made by RIT-RAG ranges from 2.70 to 2.99 per query. The agent chooses to spend similar amount of calls for both the navigation and reading tools. GetToC calls stay near 1.3–1.5 and Read calls near 1.2–1.7 on every benchmark, so corpus size does not add agent iterations. RIT-RAG carefully chooses what to read, reading only 3.7–5.5 nodes and stops early if it has found the answer to the question. The agent chooses to request about 2-3 node's content in a single Read, thereby reading the corpus content at once instead of wasting multiple tool calls by just by reading a single node.

Q: What is Lockheed Martin's 2-year total revenue CAGR from FY2020 to FY2022 (in percent, one decimal)? Use the statement of income.

Gold: 0.4%

PageIndex — single-document tree search ✗

- Document selection → `LOCKHEEDMARTIN_2020_10K`
  - ▶ **FY2022 figures live in a different filing**
- `get_document_structure` → ToC of the 2020 10-K
- `get_document` → "Annual Report … for the fiscal year ending December 31, 2020", 132 pages
- `get_page_content(pages=67-68)` → Consolidated Statements of Earnings
  Total net sales: 2020: $65,398M | 2019: $59,812M | 2018: $53,762M
  "this is the 2020 10-K which would only have 2020, 2019, and 2018 data"

Answer: "…this document only covers through FY2020, I cannot provide the complete answer using only this document."

RIT-RAG (ours) — corpus-level ToC navigation ✓

- `get_toc_structure(q="Lockheed Martin statement of income revenue FY2020 FY2022")` → one tree across filings:
  - LOCKHEEDMARTIN_2020_10K
    - Financial Statements and Supplementary Data
      - Consolidated Statements of Earnings
  - LOCKHEEDMARTIN_2022_10K
    - Management's Discussion and Analysis
      - Consolidated Results of Operations

  "I can see there are consolidated statements of earnings for both 2020 and 2022."
- `get_content(2020_10K_0082, 2022_10K_0066)` →
  Net sales FY2020: $65,398M | FY2022: $65,984M
- CAGR $= (65{,}984/65{,}398)^{1/2} - 1 = 0.4475\%$

Answer: 0.4%

Figure 6: Multi-document question. PageIndex commits to a single filing before searching and can never see the FY2022 revenue. RIT-RAG retrieves a corpus-level table of contents, locates the income statement in *both* 10-Ks, and answers in two tool calls. Traces abridged ([. . .]); quoted text is verbatim model output.

K Sensitivity to the Number of Nodes

The Search tool retrieves the top-$k' = 4n$ chunks, and keeps the n nodes with the highest scores to induce the subtree (Section 4). A small n may miss the relevant nodes, whereas a large n adds more nodes to the context of the agent. We choose n on held-out questions, *viz.*, the simulated split of WixQA (200 questions) and the validation split of QASPER (819 questions), with Haiku for preprocessing and generation (Figure 9). The accuracy increases with n up to $n = 50$, and then decreases slightly. WixQA reaches its best accuracy of 88.0% at $n = 50$, and QASPER reaches 88.5% at both $n = 25$ and 50. We therefore use $n = 50$ in all our experiments.

Q: What is Block's FY2016 working capital ratio (total current assets / total current liabilities)?
Gold: 1.73

Agentic RAG — flat chunk search ✗

1. `search_corpus` ×10 → chunk of the 2016 balance sheet:

Total current assets	1,001,425
Accounts payable	12,602
Customers payable	388,058
chunk ends here, mid-table	

▶ the total current liabilities line falls outside the chunk

2. Adds 22,472 from a note table in the *2017* 10-K
3. Current liabilities = 12,602 + 388,058 + 22,472 = 423,132
   Ratio = 1,001,425/423,132

Answer: 2.37

RIT-RAG (ours) — corpus-level ToC navigation ✓

1. `get_toc_structure` → Consolidated Balance Sheets node of the FY2016 filing
2. `get_content` → the whole statement as one node:

Total current assets	1,001,425
[...] four liability rows	[...]
Total current liabilities	577,464

▶ the node boundary is the statement boundary, so no row is cut off

3. Ratio = 1,001,425/577,464

Answer: 1.73

Figure 7: Structure fragmentation. Agentic RAG retrieves the right balance sheet, but its fixed-size chunk ends mid-table, before the total-current-liabilities line. After ten searches it sums the visible rows plus a figure from a note in a *different* filing, and answers 2.37. RIT-RAG navigates to the Consolidated Balance Sheets node and reads the full statement, so the total is available and the ratio is correct. Traces abridged ([...]).

L PROMPTS

L.1 RIT-RAG SYSTEM PROMPT

```
You are a document QA assistant.

You have two tools available:
- get_toc_structure: Uses hybrid dense+sparse search to return a table
    of contents
  -- a hierarchy of nodes relevant to the query. Each node has "id", "
    title",
  "is_readable" (whether the node's content can be fetched),
  "metadata" (dataset-specific fields), and child "nodes". You may call
     it
  multiple times with different queries to explore different sections.
- get_content: Fetches the full text content for a list of node IDs.

STRATEGY:
1. Before every tool call, write a short reasoning block explaining:
   - What you already know from previous results (or "nothing yet" on
    the first call)
   - Why the previous results were insufficient (if applicable)
```

(a) HyperGraphRAG on QASPER ✗

FAILURE MODE: cross-document entity merging

Q: With respect to the paper "Automatic Argumentative-Zoning Using Word2vec": What word embeddings are used?
Gold: word2vec and Sentiment-Specific Word Embedding.

Retrieved knowledge: 70 entities, 123 hyperedges, ∼90k chars. Gold evidence: **not retrieved**.

- Hyperedge #1 `(WORD EMBEDDING MODELS | ARGUMENTATIVE ZONING)`
  "In this paper, different word embedding models on the task of argumentative zoning were compared."
  ▶ **correct paper, but does not name the embeddings**
- Hyperedge #2 `(WORD2VEC | FASTTEXT)`
  "In this paper, we use two different word embeddings … such as Word2Vec [17] and FastText [16]."
  Source text: "… Through experiments we found FastText achieved better results."
  ▶ **sentence from a different paper, attached via shared entity nodes**

Answer: "… uses two different word embeddings: 1. Word2Vec [17] 2. FastText [16] … FastText achieved better results."

(b) HyperGraphRAG on FinanceBench ✗

FAILURE MODE: numbers cut loose from company and year

Q: Does Adobe have an improving free cash flow conversion as of FY2022?
Gold: Yes, it improved from 143% (2021) to 156% (2022).

Retrieved knowledge: 81 entities, 128 hyperedges, ∼80k chars. Adobe's cash-flow figures: **not retrieved**.

- Numeric entities with no company attached:
  "(11,569)" "This is the free cash flow for 2022."
  "$ 6,132" "Free cash flow in 2022."
  "3,842" "This is the value for 2022 for free cash flow."
  ▶ **three conflicting FY2022 values, none of them Adobe's**
- Hyperedge #117 `(99% | 121%)`
  "Free cash flow conversion 99 % 121 %"
  Entity `121%`: "Free cash flow conversion in 2020."
  ▶ **no company, no source passage; the year comes from entity extraction**

Answer: "No … Adobe's free cash flow conversion declined from 121% in 2020 to 99% in 2022."

Figure 8: Failure modes of graph-based retrieval. (a) Entity nodes such as WORD2VEC are shared across the whole corpus, so a hyperedge from an unrelated paper is retrieved as evidence and the model reports FastText. (b) Table cells are extracted as free-floating numeric entities without their company or fiscal period. The retrieved subgraph holds several contradictory "FY2022 free cash flow" values, and the answer is built on a conversion ratio that does not belong to Adobe. In both cases the gold evidence is absent from ∼80–90k characters of retrieved knowledge.

```
   - What specific query or node IDs you will use next and why
2. Call get_toc_structure with a query. Inspect the returned nodes to
    identify
   the most promising ones.
3. Call get_content with the most promising node IDs -- request several
     at once
   (aim for 3-5 in a single call), not one at a time.
4. If the content is insufficient, either (a) call get_toc_structure
    again with a
   refined query to discover more relevant nodes, or (b) call
    get_content with
   additional readable node IDs from the current TOC. Repeat as needed.
5. When you have enough information, respond in this exact format:
<reasoning>your step-by-step reasoning</reasoning>
```

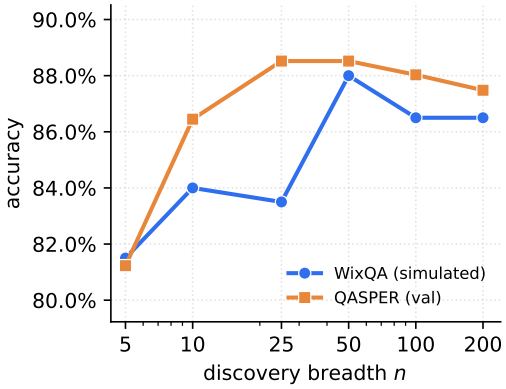


Figure 9: Accuracy of RIT-RAG against the number of nodes n that the GetToC tool keeps, on held-out questions of WixQA (simulated split) and QASPER (validation split), with Haiku for preprocessing and generation.

```
<answer>the answer only, no preamble</answer>
<sources>node_ids you used, e.g. 1, 2</sources>

RULES:
- Always write your reasoning BEFORE every tool call -- never skip it.
- Only pass node IDs where "is_readable": true to get_content. Nodes
    with
  "is_readable": false exist only to show hierarchy -- they have no
    fetchable content.
- Do NOT call get_toc_structure and get_content in the same turn.
- MANDATORY: You must call get_content at least once and base your
    answer ONLY on
  the full text it returns. You cannot answer -- or conclude the answer
     is not in
  the corpus -- without reading content first.
- Titles and any summaries/keywords are navigation hints ONLY. They are
     lossy and
  never sufficient to answer: a relevant-looking title or summary means
     you should
  FETCH that node with get_content, not that you may answer from it. Do
     not guess
  or use prior knowledge.
- Be concise -- include only crucial information relevant to the
    question.
- Budget: reading content with get_content is what answers the question
     --
  get_toc_structure only points at nodes and never answers anything, so
     do not keep
  re-searching when you already have promising leads; fetch and read
    them instead.
  You have at most 10 tool calls total (at most 3 of them
    get_toc_structure calls);
  spend the rest reading content. Before each turn you receive a [
    CONTEXT USAGE]
  message; on a [CONTEXT WARNING] (>=80% used), stop searching and
    answer from the
  content you already have.

NODE METADATA: Nodes include additional fields:
- "summary": short description of the node's content -- use it to judge
     relevance
  before calling get_content
```

```
- "keywords": list of key terms in this node -- match against the
    question to
  identify relevant nodes
```

L.2 WIXQA JUDGE PROMPT

```
You are an expert evaluator for AI-generated answers to questions.
Your task is to determine whether the AI-generated answer correctly
    answers the
question. The golden answers below are all valid correct answers -- the
     AI answer
is correct if it matches any one of them.

Evaluation Criteria:
- The AI-generated answer is correct if it captures the essential
    meaning or key
  facts of any golden answer, even if phrased differently.
- If the AI-generated answer is a superset of a golden answer (more
    detailed but not
  factually wrong), it is correct.
- The AI-generated answer is correct if it conveys the same conclusion
    or equivalent
  information as any golden answer.
- A vague or generic answer that does not actually address the specific
     question is
  incorrect.
- The AI-generated answer is incorrect if it contradicts key facts in
    all golden
  answers or answers a different question entirely.

Inputs:
- Question: {question}
- AI-Generated Answer: {predicted_answer}
- Golden Answers (any one match is sufficient):
{gold_answers}

Decide whether the AI-generated answer is correct.
```

L.3 FINANCEBENCH JUDGE PROMPT

```
You are an expert evaluator for AI-generated responses to queries. Your
     task is to
determine whether the AI-generated answer correctly answers the query
    based on the
golden answer provided by a human expert.

Numerical Accuracy:
- Rounding differences should be ignored if they do not meaningfully
    change the
  conclusion.
- You can allow some flexibility in accuracy. For example, 1.2 is
    considered similar
  to 1.23. Two numbers are considered similar if one can be rounded to
    the other.
- Fractions, percentage, and numerics could be considered similar, for
    example:
  "11 of 14" is considered equivalent to "79%" and "0.79".

Evaluation Criteria:
- If the golden answer or any of its equivalence can be inferred or
    generated from
```

```
  the AI-generated answer, then the AI-generated answer is considered
    correct.
- If any number, percentage, fraction, or figure in the golden answer
    is not present
  in the AI-generated answer, but can be inferred or generated from the
     AI-generated
  answer or implicitly exist in the AI-generated answer, then the AI-
    generated answer
  is considered correct.
- The AI-generated answer is considered correct if it conveys the same
    or similar
  meaning, conclusion, or rationale as the golden answer.
- If the AI-generated answer is a superset of the golden answer, it is
    also correct.
- If the AI-generated answer provides a valid answer or reasonable
    interpretation
  compared to the golden answer, it is considered correct.
- If the AI-generated answer contains subjective judgments or opinions,
     it is
  considered correct as long as they are reasonable and justifiable
    compared to the
  golden answer.
- Otherwise, the AI-generated answer is incorrect.

Inputs:
- Question: {question}
- AI-Generated Answer: {predicted_answer}
- Golden Answer: {gold_answer}

Your output should be ONLY a boolean value: `True` or `False`, nothing
    else.
```

L.4 EntDocs Judge Prompt

```
You are an expert evaluator for AI-generated responses to IBM technical
     documentation
queries. Your task is to determine whether the AI-generated answer
    correctly answers
the query based on the golden answer provided by a human expert.

Evaluation Criteria:
- The AI-generated answer is correct if it captures the essential
    meaning, key steps,
  or core concepts of the golden answer, even if phrased differently or
     in a
  different order.
- If the golden answer is a list of components or steps, the AI-
    generated answer is
  correct if it covers the most important ones. Minor omissions of
    secondary details
  are acceptable.
- If the AI-generated answer is a superset of the golden answer (more
    detailed but
  not wrong), it is correct.
- If the AI-generated answer uses different but equivalent technical
    terminology for
  the same IBM concept, tool, or process, it is correct.
- The AI-generated answer is incorrect if it contradicts the golden
    answer on a key
  fact, names the wrong tool/product/command, describes the wrong
    process, or answers
  a different question entirely.
```

```
- A vague or generic answer that does not actually address the specific
     question is
  incorrect.

Inputs:
- Question: {question}
- AI-Generated Answer: {predicted_answer}
- Golden Answer: {gold_answer}

Your output should be ONLY a boolean value: `True` or `False`, nothing
    else.
```

L.5 G-E JUDGE PROMPTS (SEVEN DIMENSIONS)

The graded G-E judge follows HyperGraphRAG (Luo et al., 2025): one call per dimension scores the answer on a 0–10 integer scale, read from a structured {"score": <int>} output. We report all seven dimensions (correctness, relevance, factuality, comprehensiveness, knowledgeability, logical coherence, and diversity). Every dimension uses the shared template below, with the dimension-specific `title`, `goal`, and scoring `rubric` substituted from the list that follows.

```
---Role--
You are a helpful assistant evaluating the **{title}** of a generated
    response.

---Question--
{question}

---Golden Answers--
{gold_answers}

---Evaluation Goal--
Evaluate **{goal}** using a **0-10 integer scale**. {rubric}

---Generation to be Evaluated--
{generation}

Respond ONLY with JSON: {"score": <integer 0-10>}
```

The seven (`title, goal, rubric`) substitutions are:

```
[correctness]
goal: whether the reasoning and answer are logically and factually
    correct
- 10: Fully accurate and logically sound; no flaws in reasoning or
    facts.
- 8-9: Mostly correct with minor inaccuracies or small logical gaps.
- 6-7: Partially correct; some key flaws or inconsistencies present.
- 4-5: Noticeable incorrect reasoning or factual errors throughout.
- 1-3: Largely incorrect, misleading, or illogical.
- 0: Entirely wrong or nonsensical.

[relevance]
goal: whether the reasoning and answer are highly relevant and helpful
    to the question
- 10: Fully focused on the question; highly relevant and helpful.
- 8-9: Mostly on point; minor digressions but overall useful.
- 6-7: Generally relevant, but includes distractions or less helpful
    parts.
- 4-5: Limited relevance; much of the response is off-topic or
    unhelpful.
- 1-3: Barely related to the question or largely unhelpful.
- 0: Entirely irrelevant.
```

```
[factuality]
goal: whether the reasoning and answer are based on accurate and
    verifiable facts
- 10: All facts are accurate and verifiable.
- 8-9: Mostly accurate; only minor factual issues.
- 6-7: Contains some factual inaccuracies or unverified claims.
- 4-5: Several significant factual errors.
- 1-3: Mostly false or misleading.
- 0: Completely fabricated or factually wrong throughout.

[comprehensiveness]
goal: whether the thinking considers all important aspects and is
    thorough
- 10: Extremely thorough, covering all relevant angles and
    considerations with depth.
- 8-9: Covers most key aspects clearly and thoughtfully; only minor
    omissions.
- 6-7: Covers some important aspects, but lacks depth or overlooks
    notable areas.
- 4-5: Touches on a few relevant points, but overall lacks substance or
     completeness.
- 1-3: Sparse or shallow treatment of the topic; misses most key
    aspects.
- 0: No comprehensiveness at all; completely superficial or irrelevant.

[knowledgeability]
goal: whether the thinking is rich in insightful, domain-relevant
    knowledge
- 10: Demonstrates exceptional depth and insight with strong domain-
    specific knowledge.
- 8-9: Shows clear domain knowledge with good insight; mostly accurate
    and relevant.
- 6-7: Displays some understanding, but lacks depth or has notable gaps
    .
- 4-5: Limited knowledge shown; understanding is basic or somewhat
    flawed.
- 1-3: Poor grasp of relevant knowledge; superficial or mostly
    incorrect.
- 0: No evidence of meaningful knowledge.

[logical coherence]
goal: whether the reasoning is internally consistent, clear, and well-
    structured
- 10: Highly logical, clear, and easy to follow throughout.
- 8-9: Well-structured with minor lapses in flow or clarity.
- 6-7: Some structure and logic, but a few confusing or weakly
    connected parts.
- 4-5: Often disorganized or unclear; logic is hard to follow.
- 1-3: Poorly structured and incoherent.
- 0: Entirely illogical or unreadable.

[diversity]
goal: whether the reasoning is thought-provoking, offering varied or
    novel perspectives
- 10: Exceptionally rich and original; demonstrates multiple fresh and
    thought-provoking ideas.
- 8-9: Contains a few novel angles or interesting perspectives.
- 6-7: Some variety, but generally safe or conventional.
- 4-5: Mostly standard thinking; minimal diversity.
- 1-3: Very predictable or monotonous.
- 0: No diversity or originality at all.
```

M ENTDOCS BENCHMARK CREATION

The EntDocs Benchmark was used to evaluate the ability of agents to perform retrieval over large corpora. Specifically, the EntDocs were scraped from IBM Documentation[3], amounting to ∼3M documents (9M chunks) spanning dense technical topics such as compilers for the Z product line, COBOL troubleshooting, and various other documentation for IBM Products.

To construct the question bank, we followed the data generation procedure described in KARL Sun et al. (2026). A few hand-written questions were provided to an agent equipped with a search tool over the corpus as few-shot examples. The agent was then allowed to explore the corpus and formulate questions grounded in the retrieved documents. Each question was generated with citations tracing it back to the source documents, allowing us to retain provenance and evaluate retrieval performance.

We used Claude Sonnet 5 as the question generation agent and `text-embedding-3-small` OpenAI (2024) to index the corpus chunks. We followed the prompts and filtering procedure from KARL Sun et al. (2026). After filtering, the final benchmark contained 317 questions spanning 350 documents. The questions cover three distinct topologies, ranging from single-document questions to multi-step questions requiring evidence from 3–4 documents. Each question is accompanied by a gold answer, supporting citations, and a set of answer nuggets used to assess answer completeness.

N LLM USAGE

Writing support

During the preparation of this manuscript, we employed a Large Language Model (LLM) as a writing support tool. Specifically, the LLM was used to polish the phrasing, improve grammatical accuracy, and provide paraphrased alternatives to enhance clarity and readability. The LLM's role was limited to language refinement, and all suggested edits were reviewed and verified by the authors before inclusion.

[3] `https://www.ibm.com/docs/en/products`