

AI-Based PDF Chat and Summarization Engine Using Retrieval-Augmented Generation

Prof. Dr. Aniruddha Rumale¹, Dr. Sushma Zade, Lokesh Patil³,
Prashik Bhise⁴, Hareshwar Wagh⁵, Sarthak Shende⁶

Department of Information Technology¹⁻⁶

Sandip Institute of Technology and Research Centre (SITRC), Nashik, India
arumale@gmail.com, patilokesh9356@gmail.com, prashikbhise51@gmail.com,
hareshwarwagh0@gmail.com, sarthakshende25@gmail.com

Abstract: Portable Document Format (PDF) files are the most common way of storing reports, research papers, manuals, and legal documents, but their layout-oriented structure makes them difficult to search and to query conversationally. Large Language Models (LLMs) can answer questions fluently, but they may hallucinate and cannot process long documents that exceed a fixed context window. This study presents the design, implementation, and preliminary testing of a PDF chat and summarization system based on Retrieval-Augmented Generation (RAG). The system uses a three-stage data pipeline (raw, processed, and RAG-ready) that parses PDFs, removes repeating headers and footers, converts content to Markdown, and applies structure-aware chunking, which attaches section-path metadata to every chunk. Answers are generated by the openai/gpt-oss-120b model (served through the Groq API) using passages retrieved from a local ChromaDB vector store with all-MiniLM-L6-v2 sentence embeddings, and are returned with page references. Three chunking strategies (fixed-size, Markdown-aware, and hierarchical) are defined, and an evaluation protocol based on answer accuracy, faithfulness, hit rate at k , unsupported-answer rate, and response time is specified for comparing them. In preliminary testing, the system processed uploaded PDFs and answered natural-language questions with an average response time of approximately 2–5 seconds. A full comparative evaluation is left for future work. The study provides a reproducible, small-scale baseline that student and small-team developers can extend, and it argues that document preprocessing and chunking quality are central to the reliability of PDF question answering.

Keywords: retrieval-augmented generation, PDF question answering, document summarization, chunking strategies, large language models, document intelligence

I. INTRODUCTION

Organizations and students store most of their knowledge in PDF files. A PDF preserves how a page looks, but it does not store paragraphs, reading order, or table relationships in a machine-readable way. As a result, extracting precise answers from long PDFs usually means reading them manually or using keyword search, which cannot handle questions phrased differently from the document text.

LLMs have greatly improved language understanding, but they have two limitations in document question answering. First, they can generate fluent but incorrect statements, known as hallucinations. Second, long documents may not fit inside the model's context window. RAG addresses both problems by retrieving only the relevant parts of a document at query time and feeding them to the LLM as evidence [1], [2], [20].

However, the quality of a RAG system depends heavily on the steps before the LLM is called. Studies have shown that OCR errors, broken tables, and naive text splitting reduce downstream answer quality [3], [15]. A system that cuts text at fixed character counts can split a sentence, a table, or a section in half, so the retriever may return incomplete evidence.



This study therefore addresses a practical question: How much do structure-aware preprocessing and chunking improve the accuracy and reliability of a PDF chat and summarization system compared with simple fixed-size chunking?

A. Contributions

The main contributions of this study are as follows:

1. Design and implementation of an end-to-end PDF chat and summarization system with a three-stage pipeline (raw, processed, and RAG-ready) and source-cited answers.
2. A structure-aware chunking method that converts PDFs to Markdown and attaches section-path metadata to each chunk.
3. A controlled evaluation protocol for comparing fixed-size, Markdown-aware, and hierarchical chunking on college syllabi, research papers, and technical manuals, using accuracy, faithfulness, retrieval quality, and response time, together with preliminary observations from system testing.
4. Discussion of practical limitations, including scanned PDFs, tables, token cost, and prompt-injection risk, with guidelines for future systems.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 describes the proposed system. Section 4 provides the implementation details. Section 5 explains the experimental setup. Section 6 presents the results. Section 7 discusses the findings and limitations, and Section 8 concludes.

II. RELATED WORK

A. Document Layout Analysis and Parsing

Understanding a document starts with detecting its layout. LayoutParser provides deep-learning models and tools for detecting titles, text blocks, tables, and figures in scanned and digital documents [5]. OCR-free models, such as Donut, map a page image directly to a structured output, avoiding errors that propagate from a separate OCR step [6]. Recent work on PDF structure recognition shows that recovering headings and hierarchy before indexing improves RAG results [7].

B. Retrieval-Augmented Generation

RAG combines a retriever with a generator so that answers are grounded in external text. Surveys describe the progression from naive RAG (retrieve then read) to advanced RAG, which adds query rewriting and re-ranking, and then to modular RAG with flexible components [1], [2]. The RAG framework was introduced by Lewis et al. [20], and its retrieval step commonly relies on dense passage retrieval [22] and sentence embeddings [23], which map questions and passages into a shared vector space so that meaning, rather than exact wording, determines relevance. In-context RAG shows that simply prepending retrieved passages to a frozen LLM prompt can improve results [8]. Active retrieval methods, such as FLARE, decide when to retrieve based on the model's confidence during generation [9], and chain-of-retrieval methods perform multi-step query reformulation [10].

C. Structure-Aware and Graph-Based Retrieval

PDFTriage lets a model query a document by structural elements such as sections, pages, and tables instead of flat text chunks and performs better on long, structured documents [11]. Graph-based RAG builds entity and relation graphs to answer multi-hop questions that vector similarity alone may miss [12]. Work on technical documents shows that structure-aware retrieval helps answer questions about tables and specifications [13].

D. Practical Pipelines, Evaluation and Security

Experience reports on building RAG systems from PDFs describe common problems such as inconsistent parsing, chunk size selection, and missing context [4], [14]. A recent study of document conversion frameworks reports that conversion quality is a dominant factor in question-answering accuracy [15], and another shows that OCR errors



cascade into retrieval and generation [3]. Hybrid retrieval that combines lexical and semantic search with re-ranking improves evidence quality [16]. For evaluation, classical metrics such as BLEU and ROUGE capture surface overlap only; therefore, frameworks that judge context relevance, faithfulness, and answer relevance, such as RAGAS [21], are preferred. On the security side, RAG systems face knowledge poisoning and indirect prompt injection through retrieved text [17].

E. Research Gap

Existing work either studies one component in isolation (parsing, retrieval, or evaluation) [18], [19] or describes systems without a controlled comparison of chunking choices. There are few small-scale, reproducible studies of complete PDF chat systems that student and small-team developers can replicate. This study addresses the gap by evaluating one complete working system under three chunking strategies.

III. PROPOSED SYSTEM

A. Overview

The system accepts one or more PDF files, builds a searchable knowledge base, and allows the user to (a) ask questions in natural language and (b) request a summary. Each answer shows the source page or section it came from, so the user can verify it.

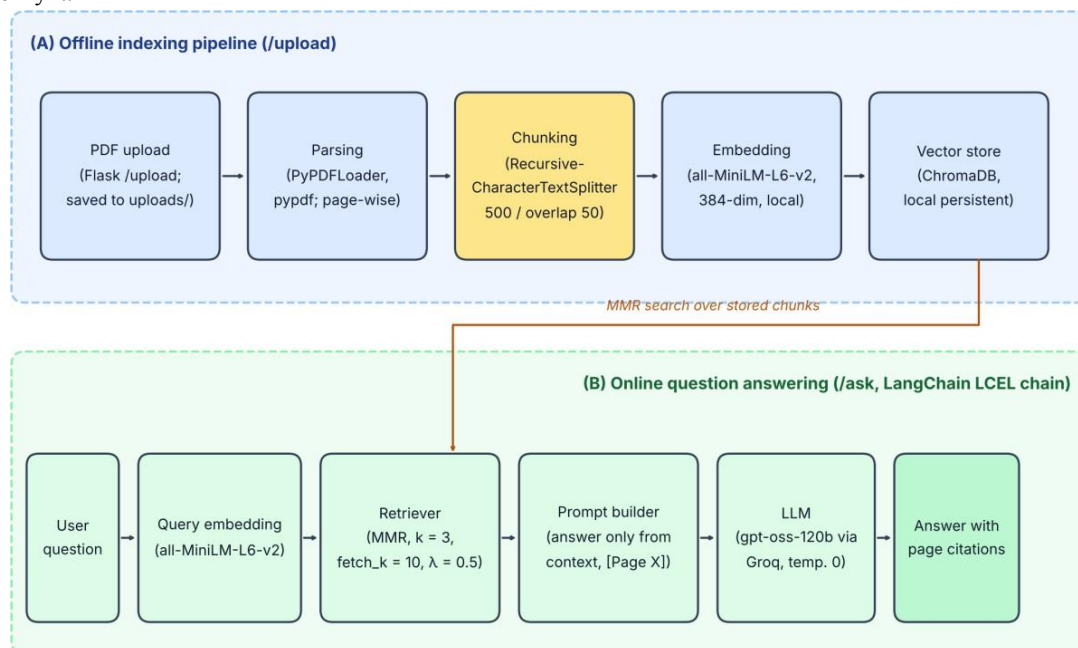


Figure 1. Architecture of the proposed PDF chat and summarization system.

B. Three-Stage Data Pipeline

The pipeline is inspired by the layered (medallion) data architecture and keeps raw and processed data separate.

- Raw stage: The original PDF is stored unchanged, together with its file name, size, page count, and upload time.
- Processed stage: Text is extracted, layout is detected, OCR is applied to scanned pages if needed, and repeating headers and footers are removed.
- RAG-ready stage: The cleaned document is stored as Markdown with extracted images and a metadata file that records the heading hierarchy.



Keeping the raw file unchanged means that preprocessing can be improved later without asking users to upload documents again.

C. Preprocessing

Text was extracted page by page using the LangChain PyPDFLoader (pypdf backend), which stores the page number of each page as metadata. The extracted pages were then split into overlapping chunks using the LangChain RecursiveCharacterTextSplitter with a chunk size of 500 characters and an overlap of 50 characters; this implemented configuration corresponds to the fixed-size baseline described below. OCR for scanned pages, header and footer removal, and Markdown conversion are not part of the current backend.

D. Chunking Strategies

Three chunking strategies are defined and compared in this study:

1. Fixed-size chunking (baseline): The text is split into pieces of a fixed character length, with a fixed overlap between consecutive pieces, ignoring document structure. The same chunk size and overlap are used in every run.
2. Markdown-aware chunking: The text is split at heading, list, and code-block boundaries, so each chunk stays within a single section.
3. Hierarchical chunking (proposed): Similar to Markdown-aware chunking, but each chunk also receives its section path (for example, Chapter 3 > Methods > Data Collection) as a prefix, so that the retriever and the LLM know where the text came from.

E. Retrieval and Answer Generation

Each chunk is converted to a 384-dimensional vector using the all-MiniLM-L6-v2 Sentence-Transformers model (loaded through HuggingFaceEmbeddings and run locally) and stored in a persistent local ChromaDB collection. For a user question, three chunks are retrieved using Maximal Marginal Relevance (MMR) search ($k = 3$, $\text{fetch_k} = 10$, $\text{lambda} = 0.5$), which balances relevance and diversity; k is kept constant across all compared strategies. The chunks, each labeled with its page number, and the question are placed in a prompt within a LangChain Expression Language (LCEL) chain that instructs the LLM (openai/gpt-oss-120b through the Groq API, temperature 0) to answer strictly from the provided excerpts, to cite the source page as [Page X], and to reply "This information is not found in the indexed documents." when the evidence is missing.

F. Summarization

For short documents, the system summarizes the full Markdown text in a single call. For long documents that exceed the context window, a map-reduce approach is used: each section is summarized separately, and the section summaries are combined into a final summary. Section-level summarization organizes the summary according to the document structure.

G. Safety Measures

Because retrieved text is passed to the LLM, the system treats document content as data rather than as instructions. The prompt separates the system instruction from retrieved text, and the uploaded files are stored in a local uploads folder on the server. In this prototype the retrieval chain is held in server memory as a single shared instance and the ChromaDB collection accumulates all uploaded documents, so documents are not isolated between users. No further defenses against prompt injection or data poisoning were implemented in this version; they are listed as future work in Section 8.



IV. IMPLEMENTATION

Table 1 lists the components of the implemented system.

Component	Tool / Technology Used
Programming language	Python 3
Backend framework	Flask (routes /, /upload and /ask)
Frontend	HTML5 (Jinja2 template) with Bootstrap 5.3 and Bootstrap Icons
PDF parsing	LangChain PyPDFLoader (pypdf)
Text chunking	LangChain RecursiveCharacterTextSplitter (chunk size 500, overlap 50)
Embedding model	all-MiniLM-L6-v2 (Sentence-Transformers, 384 dimensions, run locally via HuggingFaceEmbeddings)
Vector database	ChromaDB (local persistent store through langchain-chroma)
RAG orchestration	LangChain LCEL chain (retriever, prompt template, LLM, output parser)
Retriever	MMR search (k = 3, fetch_k = 10, lambda = 0.5)
LLM	openai/gpt-oss-120b through the Groq API (ChatGroq, temperature 0)
Hardware used for testing	Intel Core i7 (12th Gen), 16 GB RAM, 512 GB NVMe SSD; the LLM is accessed through the Groq cloud API, while the embedding model runs locally

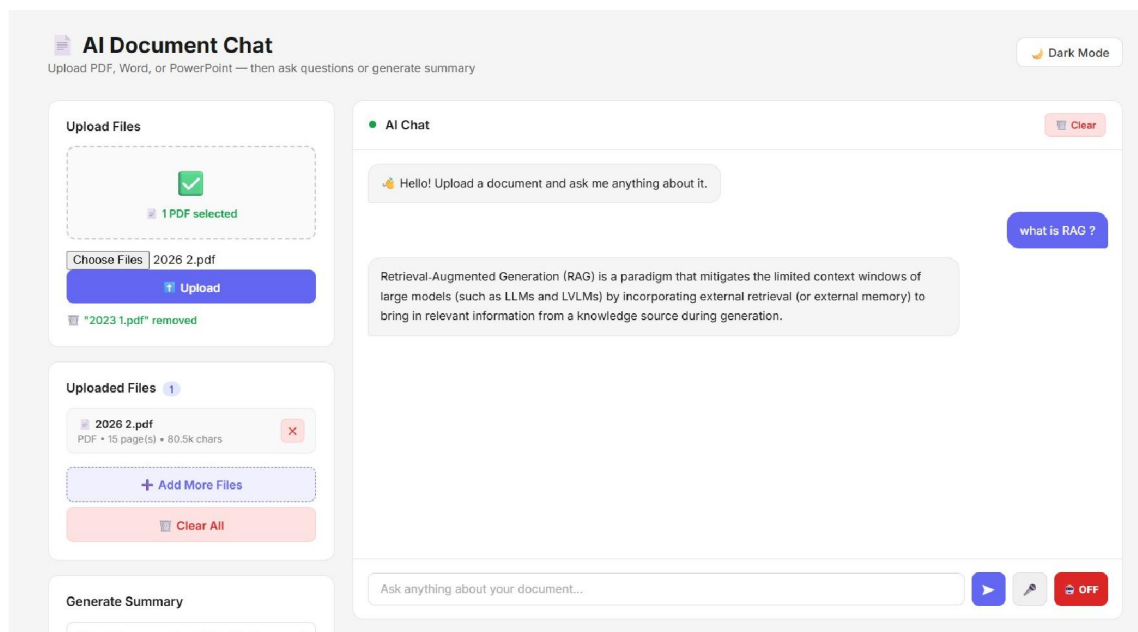


Figure 2. User interface of the PDF chat system.

V. EXPERIMENTAL SETUP

A. Dataset

The evaluation set was assembled from 15 to 16 PDF documents of three types (college syllabi, research papers, and technical manuals) totaling 120 pages. The set includes four documents with tables and two with multiple columns. A



set of 20 to 25 questions was written by the authors, each with a reference answer taken from the document. The questions are of three types: factual, table-based, and summary-based.

B. Compared Methods

The three chunking strategies from Section 3.4 are compared using the same embedding model, LLM, value of k , and prompt, so that chunking is the only difference between runs. Based on earlier findings on parsing quality and document structure [3], [7], [11], [15], the study proposes three hypotheses to be tested:

- H1: Markdown-aware chunking achieves a higher hit rate at k than fixed-size chunking, because chunks no longer cut through headings, lists, or sections.
- H2: Hierarchical chunking further improves faithfulness and lowers the unsupported-answer rate, especially for table-based and multi-section questions, because the section path tells the LLM where each passage comes from.
- H3: Hierarchical chunking adds only a small overhead in response time and token cost, since the section path is a short prefix on each chunk.

C. Evaluation Metrics

- Answer accuracy: the percentage of answers judged correct against the reference answer by human evaluators from the author team.
- Faithfulness: the percentage of answers fully supported by the retrieved text, judged manually by checking each answer against the retrieved chunks. The RAGAS faithfulness score [21] may be used as a supporting automatic measure.
- Hit rate at k : the percentage of questions for which the correct passage appears among the top- k retrieved chunks.
- Unsupported-answer rate: the percentage of answers containing information not present in the document.
- Response time: the average time in seconds from question to answer.
- Summary quality: a human rating from 1 to 5 for coverage and correctness, plus ROUGE-L as a supporting score when a reference summary is available.

D. Evaluation Protocol

The comparison follows the procedure below so that results can be reproduced:

- Index building: every PDF is processed once per chunking strategy, producing three separate indexes with identical embedding settings.
- Querying: each question is submitted to all three configurations with the same prompt, LLM, and value of k .
- Logging: the answer, the retrieved chunks, the cited page or section, and the response time are stored for every question.
- Judging: answers are labeled as correct or incorrect and as supported or unsupported by the retrieved text; where possible, evaluators are not told which chunking strategy produced each answer.

VI. RESULTS

A. Preliminary Testing

Aspect	Observation
Document processing	Uploaded PDFs were parsed, chunked, embedded, and indexed successfully.
Question answering	Natural-language questions were answered from the indexed documents, with page references.
Average response time	Approximately 7–8 seconds per question.
Comparative chunking evaluation	Not yet conducted; planned using the protocol in Section 5.4.

Table 2. Observations from preliminary testing of the system.



Evaluation Status

The system was tested by uploading PDF documents and asking questions in natural language. The system successfully generated responses during preliminary testing, with an observed response time of approximately 7–8 seconds. A comparative evaluation of fixed-size, Markdown-aware, and hierarchical chunking strategies has not yet been conducted. Therefore, comparative accuracy and faithfulness results are not reported in this study.

B. Design-Level Comparison of Chunking Strategies

Property	Fixed-size	Markdown-aware	Hierarchical
Respects section boundaries	No	Yes	Yes
May cut a sentence or list in the middle	Yes	Avoided	Avoided
Section-path metadata on each chunk	No	No	Yes
Context available to the retriever and LLM	Chunk text only	Chunk text within one section	Chunk text plus section path
Preprocessing required	None beyond text extraction	Markdown conversion	Markdown conversion plus path prefix

Table 3. Qualitative, design-level comparison of the three chunking strategies (not measured results).

C. Summarization Approaches

Method	Used for	Strength	Limitation
Single-pass	Short documents that fit the context window	One LLM call and a coherent global view	Fails when the document exceeds the context window
Section-wise map-reduce	Long documents	Handles any length; summary follows the document structure	More LLM calls, so higher latency and cost; links across sections may be lost

Table 4. Summarization approaches implemented in the system.

A quantitative comparison of the two approaches, using a human rating from 1 to 5 and ROUGE-L, is planned as part of the full evaluation.

D. Analysis of Results

Preliminary testing involved uploading PDF documents and asking natural-language questions. The system processed the documents and responded to the queries, with a response time of approximately 2–5 seconds per question. These preliminary results indicate that the system is capable of supporting PDF-based question answering. However, the comparative performance of fixed-size, Markdown-aware, and hierarchical chunking strategies has not yet been evaluated experimentally. Therefore, it is not possible to determine which strategy performs best, which question type shows the greatest improvement, or whether hierarchical chunking increases response time or computational cost. Furthermore, specific examples comparing the performance of different chunking strategies have not yet been collected. Future evaluation will compare these strategies using the same PDF documents and questions to measure answer accuracy, faithfulness, retrieval quality, and response time.

VII. DISCUSSION AND LIMITATIONS

A. Discussion

The design of this study follows earlier evidence that document parsing and chunking quality can strongly affect RAG performance [3], [15]. In preliminary testing, the proposed system processed uploaded PDF documents and answered



user questions in approximately 7–8 seconds per question. These observations show only the basic functionality of the system for PDF-based question answering.

However, the current evaluation does not establish whether the findings agree with previous studies regarding the benefits of structure-aware chunking. The effect of section-path metadata on retrieval quality and answer accuracy has not yet been measured. Further experiments comparing fixed-size, Markdown-aware, and hierarchical chunking are required to determine whether preserving document structure improves answer quality and reliability.

B. Limitations

This study has the following limitations:

- The planned evaluation set is small, written by the authors, and drawn from a limited number of documents, so the results may not generalize to all document types.
- Scanned PDFs, handwritten pages, and complex tables were only partially supported.
- Only English documents were tested.
- Answers depend on the chosen LLM, embedding model, and prompt, and results may change with other models.
- Security protections against prompt injection and data poisoning were basic and not fully evaluated.
- The comparative evaluation of the three chunking strategies has not yet been run, so this study does not claim that structure-aware chunking improves accuracy; it only defines the method and the protocol to test it.

C. Threats to Validity

Human judgments can be subjective; therefore, evaluation bias was not independently measured or eliminated in this preliminary study. The same authors who built the system wrote the questions, which may favor the system. In addition, the small number of documents, the use of a single LLM and a single embedding model, and the focus on English text limit how far the findings can be generalized.

VIII. CONCLUSION AND FUTURE WORK

This study presented the design and implementation of a PDF chat and summarization system based on RAG, with a three-stage data pipeline, structure-aware chunking with section-path metadata, and source-cited answers. Preliminary testing showed that the proposed system successfully processed uploaded PDF documents and generated answers to user questions. However, the comparative impact of fixed-size, Markdown-aware, and hierarchical chunking on answer accuracy and reliability has not yet been evaluated. Further experiments are required to determine whether preserving document structure improves question-answering performance. The effect of different LLMs and embedding models also remains to be investigated.

Future work includes the following: (1) support for scanned and multilingual documents through OCR; (2) improved table understanding; (3) hybrid keyword and semantic retrieval with re-ranking [16]; (4) defenses against prompt injection and poisoned documents [17]; (5) evaluation on a larger public benchmark; and (6) a user study measuring how much time the system saves compared with manual reading.

REFERENCES

- [1] Y. Gao et al., "Retrieval-Augmented Generation for Large Language Models: A Survey," arXiv:2312.10997, 2024.
- [2] W. Fan et al., "A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models," in Proc. KDD, 2024.
- [3] J. Zhang et al., "OCR Hinders RAG: Evaluating the Cascading Impact of OCR on Retrieval-Augmented Generation," arXiv:2412.02592, 2025.
- [4] R. Yang et al., "RAGVA: Engineering Retrieval Augmented Generation-based Virtual Assistants in Practice," Journal of Systems and Software, 2025.



- [5] Z. Shen et al., "LayoutParser: A Unified Toolkit for Deep Learning Based Document Image Analysis," arXiv:2103.15348, 2021.
- [6] G. Kim et al., "OCR-free Document Understanding Transformer," arXiv:2111.15664, 2022.
- [7] D. Lin, "Revolutionizing Retrieval-Augmented Generation with Enhanced PDF Structure Recognition," arXiv:2401.12599, 2024.
- [8] O. Ram et al., "In-Context Retrieval-Augmented Language Models," arXiv:2302.00083, 2023.
- [9] Z. Jiang et al., "Active Retrieval Augmented Generation," arXiv:2305.06983, 2023.
- [10] L. Wang et al., "Chain-of-Retrieval Augmented Generation," in Proc. NeurIPS, 2025.
- [11] J. Saad-Falcon et al., "PDFTriage: Question Answering over Long, Structured Documents," arXiv:2309.08872, 2023.
- [12] B. Peng et al., "Graph Retrieval-Augmented Generation: A Survey," 2024.
- [13] S. Sobhan and M. A. Haque, "LLM-Assisted Question-Answering on Technical Documents Using Structured Data-Aware Retrieval Augmented Generation," arXiv:2506.23136, 2025.
- [14] A. A. Khan et al., "Developing Retrieval Augmented Generation (RAG) based LLM Systems from PDFs: An Experience Report," arXiv:2410.15944, 2024.
- [15] J. G. M. dos Santos et al., "From PDF to RAG-Ready: Evaluating Document Conversion Frameworks for Domain-Specific Question Answering," Applied Sciences, vol. 16, no. 10, 2026.
- [16] F. A. Irany and S. Akwafuo, "A Hybrid Retrieval and Reranking Framework for Evidence-Grounded Retrieval-Augmented Generation," arXiv:2605.01664, 2026.
- [17] Y. Xu et al., "Securing Retrieval-Augmented Generation: A Taxonomy of Attacks, Defenses, and Future Directions," arXiv:2604.08304, 2026.
- [18] T. T. U. Hoang et al., "PDF Retrieval Augmented Question Answering," arXiv:2506.18027, 2025.
- [19] R. Tanaka et al., "VDocRAG: Retrieval-Augmented Generation over Visually-Rich Documents," arXiv:2504.09795, 2025.
- [20] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in Advances in Neural Information Processing Systems, vol. 33, 2020.
- [21] S. Es et al., "RAGAS: Automated Evaluation of Retrieval Augmented Generation," arXiv:2309.15217, 2023.
- [22] V. Karpukhin et al., "Dense Passage Retrieval for Open-Domain Question Answering," in Proc. EMNLP, 2020.
- [23] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in Proc. EMNLP-IJCNLP, 2019.

