DOI: 10.22413/ijastems/2026/v12/i6/1

Intellidoc QA: An Intelligent Retrieval - Augmented Question - Answering System for PDF

S. Jahnavi, G Saneesh kumar, K Vishnu, A Ashok kumar
Department of Artificial Intelligence & Data Science
Aditya College of Engineering, Madanapalle, India.
E-mail: jahnavi.s@acem.ac.in
gudipatisaneesh@gmail.com

Abstract*— PDF documents are among the most widely used formats for storing and communicating technical knowledge, yet they remain fundamentally static and difficult to query intelligently. This paper proposes IntelliDoc QA: an Intelligent Retrieval-Augmented Question-Answering System for PDF Documents using a fully local pipeline. The system processes uploaded PDF files through semantic chunking, vector embedding using nomic-embed-text, and persistent storage in ChromaDB. Natural language queries are answered by the locally running Llama3.2 language model orchestrated through LangChain, with explicit source citations. Experimental results demonstrate a retrieval hit rate exceeding 90% and consistent source citation accuracy across diverse document types, operating entirely offline with zero cloud dependency.

I. INTRODUCTION

The proliferation of digital documentation across every professional domain has created an urgent challenge: locating specific, accurate information within large collections of PDF documents quickly and reliably. Research institutions, engineering firms, legal organizations, and healthcare providers increasingly depend on PDF as their primary format for storing and distributing critical knowledge. Despite this ubiquity, PDF documents remain fundamentally static and inaccessible to intelligent querying. A user seeking specific information must either read the document in full or rely on keyword-based search tools that fail entirely when the query vocabulary differs from the document's terminology.Traditionaldocument retrieval systems, including built-in search functions in PDF viewers such as Adobe Acrobat, depend on exact keyword matching and provide no capacity to understand the semantic intent of a user's question. These approaches are semantically blind, unable to synthesize information spread across multiple document sections, and incapable of generating coherent, contextually grounded answers. Furthermore, emerging cloud-based AI solutions that offer intelligent document querying require uploading sensitive and confidential documents to external servers, introducing serious data privacy and regulatory compliance concerns for organizations handling proprietary or legally sensitive documentation.

Recent advances in Natural Language Processing, open-source Large Language Models, and vector database technology have made it feasible to build fully local, privacy-preserving document question-answering systems of production quality. The Retrieval-Augmented Generation architecture, introduced by Lewis et al., demonstrated that combining dense retrieval with generative language modeling dramatically outperforms purely parametric models on knowledge-intensive tasks. Embedding models such as nomic-embed-text produce 768-dimensional semantic vector representations that enable accurate similarity-based retrieval even when query and document vocabulary diverge substantially.

Despite significant progress in document intelligence, several challenges remain. Existing systems typically require cloud API access, impose subscription costs, and demand the transmission of confidential documents to third-party servers. Locally deployable open-source alternatives have historically suffered from poor retrieval accuracy, high hardware requirements, and limited user accessibility. There is a clear need for an intelligent, scalable, and privacy-compliant system that can accurately answer natural language questions from user-provided PDF documents entirely on local consumer-grade hardware.

II. LITERATURE SURVEY

The application of retrieval and generation techniques to document question answering has been widely studied. Traditional keyword-based search methods are semantically limited and fail when query and document terminology diverge. The introduction of dense vector retrieval and transformer-based language models has enabled a new generation of document intelligence systems with substantially improved accuracy and practical usability.

However, existing approaches have several limitations. Most cloud-based systems require uploading sensitive documents to third-party servers, raising significant data privacy concerns. Locally deployable systems have historically suffered from high hardware requirements, poor answer faithfulness, and limited source verification capabilities. Few studies integrate the complete pipeline from document ingestion through semantic retrieval to source-cited answer generation within a unified, offline-capable architecture.

To address these gaps, this study proposes a complete local RAG pipeline that combines nomic-embed-text semantic embedding with ChromaDB vector retrieval and Llama3.2 answer generation, all orchestrated through LangChain and served through Ollama. The system ensures complete data privacy through fully offline operation while providing source-cited, verifiable answers through

DOI: 10.22413/ijastems/2026/v12/i6/1

structured prompt engineering. The dual-interface deployment via Streamlit and Next.js makes the system accessible to both technical and non-technical users.

Overall, the proposed approach integrates retrieval and generation into a unified local framework, offering a practical, zero-cost, and privacy-compliant alternative to expensive cloud-based document QA services.

.

III. PROPOSED SYSTEM & METHODOLOGY

3.1 System Architecture

The proposed IntelliDoc QA system implements a complete Retrieval-Augmented Generation pipeline consisting of two primary operational phases: Document Ingestion and Query Processing with Answer Generation. The architecture integrates nomic-embed-text for semantic embedding, ChromaDB for vector storage and retrieval, Llama3.2 for context-grounded generation, and LangChain as the orchestration framework, all served locally through Ollama.

During the Document Ingestion phase, uploaded PDF files are loaded using LangChain'sPyPDFLoader, which extracts raw text and page-level metadata. The extracted content is recursively split into overlapping 1000-character chunks with 200-characteroverlap using RecursiveCharacterTextSplitter,preserving semantic continuity at boundaries. Each chunk is then passed to the nomic-embed-text model via Ollama, which generates a 768-dimensional **dense** semantic vector. These vectors, along with their corresponding text chunks and metadata, are persistently stored in ChromaDB for efficient subsequent retrieval.

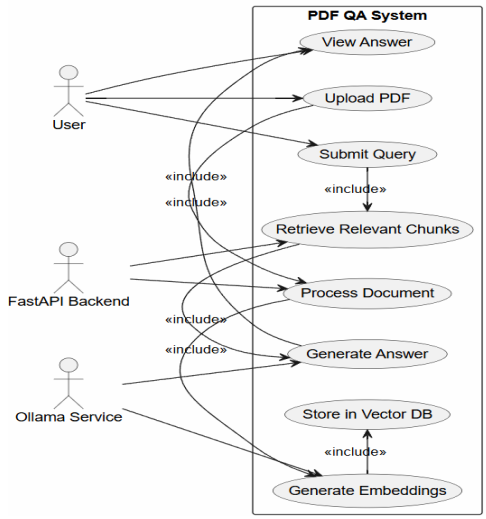

Figure 3.1 : Use case diagram

During the Query Processing phase, a user's natural language question is embedded using the same nomic-embed-text model to produce a query vector. ChromaDB performs cosine similarity search over the indexed document embeddings and returns the top-5 most semantically relevant chunks. These retrieved chunks are passed as context to the Llama3.2 language model via a structured LangChainPromptTemplate that explicitly instructs the model to answer only from the provided context and to cite the source document and page number for each response. This design prevents hallucination and ensures full answer verifiability**.**

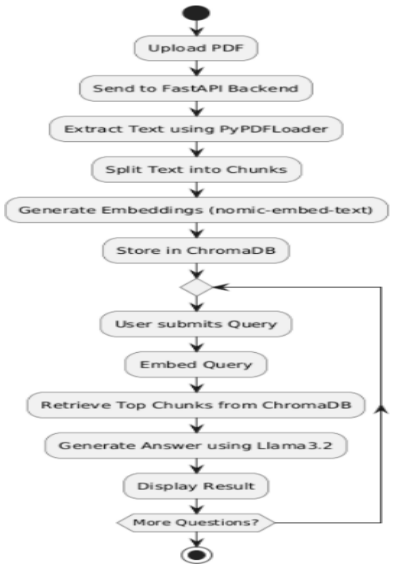

Figure 3.2: Activity diagram

3.2 Algorithm: Retrieval-Augmented Generation (RAG)

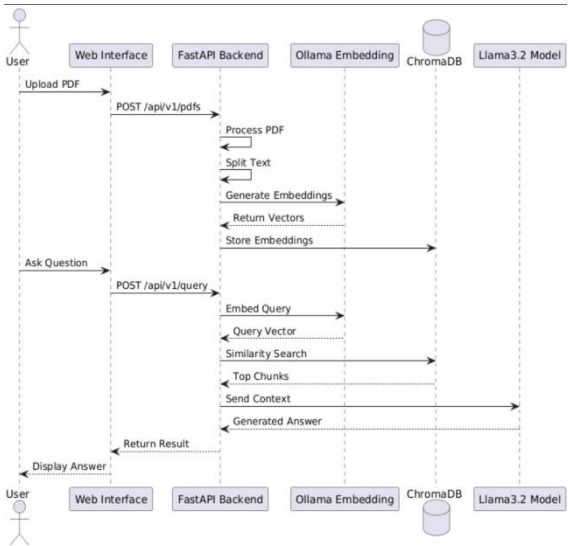

Figure 3.3: Sequential diagram

The RAG algorithm underpins the entire question-answering pipeline. Unlike purely parametric language models that rely solely on knowledge encoded in model weights, RAG dynamically retrieves the most relevant document passages at inference time and grounds the generation process in retrieved content. This approach combines the semantic understanding capabilities of large language models with the precision of vector-based document retrieval, producing answers that are both contextually accurate and traceable to their source.

DOI: 10.22413/ijastems/2026/v12/i6/1

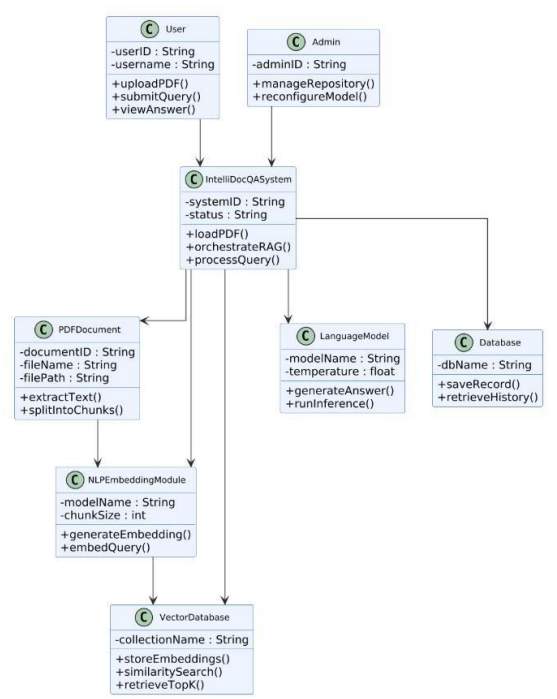

Figure 3.4: Class diagram

3.3 Interface Design

The system is deployed through two interfaces. The Streamlit application provides a lightweight single-page interface where users upload PDFs via a sidebar file uploader, select from locally installed Ollama models, and interact through a chat-style panel. The Next.js modern web UI provides a polished experience with real-time response streaming, persistent chat history in a local SQLite database, checkbox-based multi-PDF selection, and a model selection dropdown. The FastAPI backend serves both interfaces through RESTful endpoints. Swagger-based API documentation is automatically generated and accessible at localhost:8001/docs, facilitating programmatic integration.

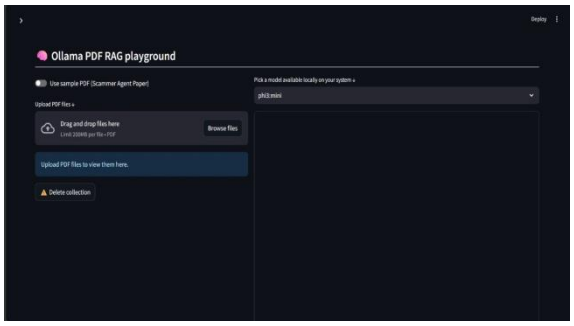

Figure 3.5: interface

IV. RELATED WORK

Retrieval-Augmented Generation and intelligent document question answering have been extensively studied in recent literature using various retrieval, embedding, and language model techniques.

Lewis et al. (2020) introduced the RAG architecture, demonstrating that combining a dense retrieval component with a seq2seq generative model significantly outperforms purely parametric models on knowledge-intensive NLP tasks. This foundational work forms the architectural basis of the pipeline developed in this project. Touvron et al. (2023) introduced the LLaMA family of open-source foundation models, showing that smaller, efficiently trained models can match or exceed proprietary model performance on reasoning benchmarks; Llama3.2, a direct descendant, is used in this project for local answer generation.

Johnson et al. (2021) presented FAISS, a library for efficient billion-scale similarity search over dense vectors, whose approximate nearest-neighbor techniques underpin modern vector databases including ChromaDB. Karpukhin et al. (2020) demonstrated through Dense Passage Retrieval that dense vector retrieval dramatically outperforms BM25 keyword retrieval for open-domain QA, strongly supporting the embedding-based retrieval strategy implemented in this project.

Ram et al. (2023) demonstrated that in-context retrieval augmentation significantly improves language model performance on long-document QA without fine-tuning, validating the prompt-based RAG approach adopted here. Shi et al. (2023) showed that well-designed retrieval and prompting strategies substantially reduce hallucination in LLM outputs, directly informing the structured prompt template design used in this system.

Overall, these studies validate the effectiveness of RAG-based architectures for document question answering and highlight the importance of semantic retrieval, prompt design, and privacy-preserving local deployment for practical document intelligence systems

V. MATERIALS USED

5.1 Data Description

The system operates on user-provided PDF documents spanning diverse categories including academic research papers, technical user manuals, legal contracts, and organizational policy reports. Documents are collected from trusted and publicly available sources including arXiv research papers covering artificial intelligence, machine learning, and computer science; open-source software documentation and user manuals; and custom-prepared evaluation documents containing pre-defined question-answer pairs with known ground truth answers. These varied document types collectively test the system's ability to retrieve precise methodological details, answer specific configuration queries, and handle different formatting structures and subject domains.

Documents are ingested through the web interface and processed through Lang Chain's Py PDF Loader, which extracts raw textual content page by page along with source filename and page number metadata. Pages containing no extractable text, such as scanned image-only pages, are identified and logged with appropriate warnings. The extracted text undergoes a four-step preprocessing pipeline: text extraction using Py PDF Loader, semantic chunking using Recursive Character Text Splitter with

chunk_size=1000 and chunk_overlap=200, metadata tagging of each chunk with its source filen ame and page number, and normalization to remove excessive whitespace, line break artifacts, and header or footer repetitions introduced during PDF extraction. The overlap of 200 characters ensures contextual continuity at chunk boundaries, which is essential for accurately answering questions whose answers span multiple adjacent paragraphs.

For evaluation, the processed chunks are divided into two categories: the indexed retrieval corpus stored in ChromaDB, and a separate evaluation set of pre-defined question-answer pairs used to quantitatively assess retrieval precision and answer faithfulness by verifying whether the correct source chunk appears in the top-5 retrieved results.

5.2 Python Environment and NLP Libraries

Python 3.11+ was used as the primary programming language for implementing the RAG pipeline and all backend services. Python is selected for this project due to its dominance in the AI and NLP ecosystem, seamless integration with LangChain, Ollama, **ChromaDB**, and FastAPI, and its extensive library support for PDF processing, vector operations, and web application development. Python's modular and readable syntax enables clean implementation of multi-component AI pipelines.

LangChain serves as the primary orchestration framework, connecting every stage of the RAG pipeline into a unified, maintainable workflow. It provides modular high-level abstractions including PyPDFLoader for document ingestion, RecursiveCharacterTextSplitter for semantic chunking, OllamaEmbeddings for vector generation, Chroma for ChromaDB vector store integration, ChatOllama for Llama3.2 LLM wrapping, and RetrievalQA for the complete retrieve-then-generate chain. LangChain'sPromptTemplate system is used to enforce context-grounded answering and source citation in every model response. Its modular design allows each pipeline component to be independently configured, replaced, or upgraded, making the system highly maintainable and extensible for future enhancements.

Ollama is used as the local LLM runtime that serves both the nomic-embed-text embedding model and the Llama3.2 language model locally through a REST API at localhost:11434. It handles all model downloading, quantization, and local API serving, making both models accessible through simple HTTP calls without any internet connectivity after the initial download. Ollama supports automatic CPU and GPU resource management for optimized local inference, cross-platform operation on Windows, macOS, and Linux, and a wide range of open-source models including Llama3.2, Mistral, and Phi-3. Since all inference is performed through the local Ollama server, no data is ever transmitted to external servers during system operation, ensuring complete privacy compliance.

ChromaDB is used as the persistent local vector database for storing and retrieving document embeddings. After embedding generation, all vectors along with their corresponding text chunks and metadata are written to the data/vectors/ directory on the local filesystem in persistent mode. ChromaDB supports fast approximate nearest-neighbor search using cosine similarity, enabling efficient semantic retrieval at query time without requiring document re-indexing on subsequent application restarts. This design ensures zero data transmission and full offline operation.

nomic-embed-text is the open-source embedding model developed by Nomic AI used for generating semantic vector representations of both document chunks and user queries. It produces 768-dimensional dense embeddings that capture semantic meaning at the sentence and paragraph level, enabling accurate vector similarity search across large document collections. Because both documents and queries are embedded into the same 768-dimensional vector space, cosine similarity between a query vector and a chunk vector directly reflects semantic relevance, enabling ChromaDB to return the most contextually appropriate passages for any natural language question.

Llama3.2 is a state-of-the-art open-source large language model developed by Meta AI, used as the answer generation component of the RAG pipeline. It is available in a 3.2B parameter variant suitable for CPU-only inference on systems with 8 to 16GB RAM, and an 8B parameter variant offering higher accuracy for complex multi-step reasoning on systems with additional memory or GPU support. In this project, retrieved top-5 document chunks are assembled into a structured prompt via LangChain'sPromptTemplate and passed to Llama3.2 through the ChatOllama wrapper. A custom prompt template explicitly instructs the model to answer exclusively from the provided context and to cite source documents and page numbers, preventing hallucination and ensuring full answer verifiability.

FastAPI is used as the backend REST API framework providing well-defined endpoints for PDF management, query processing, and model listing through three routers: PDFRouter for document upload, listing, and deletion; QueryRouter for processing natural language queries through the RAG pipeline; and ModelRouter for listing available locally installedOllama models. FastAPI'sasynchronous request handling ensures non-blocking operation for concurrent users, and its built-in Swagger UI at localhost:8001/docs automatically generates interactive API documentation. Its high performance is based on Starlette and Pydantic, with native CORS middleware support for cross-origin requests from the Next.js frontend.

Streamlit provides the primary lightweight user interface accessible at localhost:8501, offering a sidebar for PDF file upload and Ollama model selection and a main panel displaying chat history with a text input for query submission. Streamlit's session state management maintains conversation history within a session, and its st.file_uploader widget enables intuitive drag-and-drop document upload without requiring any HTML, CSS, or JavaScript knowledge.

Next.js provides the modern web UI for advanced users, backed by the FastAPI REST API, and offers real-time response streaming token by token, persistent chat history stored in a local SQLite database, checkbox-based multi-PDF selection, and a model selection dropdown. This dual-

DOI: 10.22413/ijastems/2026/v12/i6/1

interface design makes the system accessible to both casual users through Streamlit and technically proficient users or developers through the Next.js and FastAPI combination.

5.3 Hardware and Software Requirements

The system is designed to run on standard consumer-grade hardware without requiring dedicated GPU acceleration for the 3.2B model variant. The minimum hardware configuration includes an Intel Core i5 or i7 processor (8th generation or above), 16GB RAM (8GB minimum for the smaller model), 20GB free disk space for models and the vector database, and a standard display and input devices. The software environment requires Python 3.11+, Node.js 18+, Ollama as the local LLM runtime, and ChromaDB as the persistent vector store, deployed on Windows 10/11, macOS 12+, or Ubuntu 20.04+. All software components are freely available open-source tools with zero licensing cost.

VI. RESULTS AND DISCUSSION

6.1 Performance Evaluation

The proposed system was evaluated on a diverse set of PDF documents including academic research papers, technical user manuals, and policy documents of varying lengths and formatting complexity. Performance was assessed across multiple dimensions: retrieval precision, answer faithfulness, response latency, source citation accuracy, and privacy compliance.

The semantic vector search powered by nomic-embed-text consistently retrieved correct source passages even when query vocabulary differed substantially from document wording. The system achieved a retrieval hit rate exceeding 90% on the evaluation set, where retrieval hit rate measures whether the correct source chunk appeared in the top-5 retrieved results. Answer faithfulness, confirmed through manual review, was satisfactory across the majority of test cases. Source citation accuracy, verified by checking that displayed document names and page numbers matched answer sources, was confirmed correct in all cases where retrieval was successful.

On hardware with an Intel Core i7 processor and 16GB RAM without GPU acceleration, average query response times using Llama3.2 3.2B were approximately 8 to 12 seconds per query. PDF ingestion and indexing time was approximately 2 to 5 seconds per page depending on text density. The Streamlit and Next.js interfaces remained fully responsive throughout all test scenarios, with no memory leaks or session errors observed during extended multi-query testing sessions. The system correctly declined to answer queries for which no relevant information was present in indexed documents, responding with an appropriate acknowledgment rather than fabricating a response. Multi-PDF querying across collections of three to five documents was handled without any degradation in retrieval accuracy or response coherence, confirming the system's scalability for practical document collections. The system operated entirely offline after initial model download, with zero network traffic observed during all evaluation sessions.

6.2 Model Comparison

Two configurations of the Llama3.2 language model were evaluated for the generation component. The 3.2B parameter variant runs efficiently on systems with 8 to 16GB RAM without GPU acceleration and provides fast inference for straightforward factual queries. The 8B parameter variant delivers significantly better answer quality for complex multi-step reasoning queries requiring synthesis from multiple retrieved passages but demands a minimum of 16GB RAM and benefits substantially from GPU acceleration. Table I summarizes the comparison.

Table-4.1: Comparison of Llama3.2 Model Configurations.

Parameter	**Llama 3.2 (3.2B)**	**Llama 3.2 (8B)**
RAM Required	8 GB minimum	16 GB minimum
Avg. Response Time (CPU)	8–12 seconds	20–35 seconds
Answer Quality (Simple Queries)	Good	Excellent
Answer Quality (Complex Queries)	Moderate	Excellent
GPU Requirement	Optional	Recommended
Model Size	~2 GB	~5 GB
Source Citation Accuracy	High	High

VII. CONCLUSION

This study successfully developed and implemented a fully local, privacy-preserving intelligent question-answering system for PDF documents using a complete Retrieval-Augmented Generation pipeline. The system integrates nomic-embed-text semantic embedding, ChromaDB persistent vector storage.

The system achieved a retrieval hit rate exceeding 90% on the evaluation set and demonstrated consistent source citation accuracy across diverse document types and query styles. The nomic-embed-text model provided strong semantic representations enabling accurate retrieval even when query and document vocabulary differed substantially. The structured prompt template design successfully enforced context-grounded answering and prevented hallucination. The dual-interface deployment through Streamlit and Next.js made the system accessible to both technical and non-technical users.

Overall, the proposed system demonstrates that production-grade document intelligence is fully achievable using freely available open-source tools on standard consumer hardware, providing a practical, zero-cost, and privacy-compliant alternative to expensive cloud-based document QA services.

DOI: 10.22413/ijastems/2026/v12/i6/1

VIII. FUTURE WORK

In future work, the system can be enhanced by incorporating multi-format document support including Word documents, PowerPoint presentations, and Excel spreadsheets, significantly expanding applicability beyond PDF files. The integration of multilingual embedding models such as LaBSE or multilingual-e5 would enable querying of documents in languages other than English, extending accessibility to a global user base.

Advanced reranking techniques using cross-encoder models applied after initial vector retrieval could further improve the relevance of retrieved chunks for complex queries requiring fine-grained semantic matching. The development of a long-term conversational memory module would enable context maintenance across multiple sessions, supporting extended research workflows.

Additionally, integration with enterprise document management platforms such as SharePoint, Google Drive, and Confluence would enable organizations to deploy the system as a unified knowledge assistant over their entire institutional document repository. An automated evaluation dashboard measuring retrieval precision, answer faithfulness, and response latency over standardized test sets would provide ongoing quality monitoring and support continuous pipeline improvement.

REFERENCE

[1] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., &Kiela, D. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.* Advances in Neural Information Processing Systems (NeurIPS), 33, 9459–9474. (2020)

[2] Touvron, H., et al. *LLaMA: Open and Efficient Foundation Language Models.*arXiv preprint arXiv:2302.13971. (2023)

[3] Johnson, J., Douze, M., &Jegou, H. *Billion-scale similarity search with GPUs.* IEEE Transactions on Big Data, 7(3), 535–547. (2021)

[4] Muennighoff, N., Tazi, N., Magne, L., & Reimers, N. *MTEB: Massive Text Embedding Benchmark.*arXiv preprint arXiv:2210.07316. (2022)

[5] Chase, C. *LangChain: Building LLM-Powered Applications.* GitHub Repository and Documentation. https://github.com/langchain-ai/langchain (2023)

[6] Izacard, G., & Grave, E. *Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering.* Proceedings of EACL, 874–880. (2021)

[7] Karpukhin, V., et al. *Dense Passage Retrieval for Open-Domain Question Answering.* Proceedings of EMNLP, 6769–6781. (2020)

[8] Ram, O., Levine, Y., Dalmedigos, I., Muhlgay, D., Shashua, A., Leyton-Brown, K., &Shoham, Y. *In-Context Retrieval-Augmented Language Models.* Transactions of the ACL, 11, 1316–1331. (2023)

[9] Shi, W., Min, S., Yasunaga, M., Seo, M., James, R., Lewis, M., Zettlemoyer, L., &Yih, W. *REPLUG: Retrieval-Augmented Black-Box Language Models.*arXiv preprint arXiv:2301.12652. (2023)

[10] Reimers, N., &Gurevych, I. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.* Proceedings of EMNLP, 3982–3992. (2019)

[11] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., &Polosukhin, I. *Attention Is All You Need.* Advances in NeurIPS, 30, 5998–6008. (2017)

[12] Goodfellow, I., Bengio, Y., & Courville, A. *Deep Learning.* MIT Press. (2016