International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 04 | Apr 2026 www.irjet.net p-ISSN: 2395-0072

Enterprise Document Intelligence: A Production-Grade RAG Pipeline with Hybrid Retrieval, Reranking, and Conversational Memory

Adeeb Sailani Shaikh[1], Sanidhya Sachin Kulkarni[2], Ratnamala Kumar Sudhir Paswan[3]

[1]Student, Pune Institute of Computer Technology (PICT), Pune, India
[2]Student, Pune Institute of Computer Technology (PICT), Pune, India
[3]Assistant Professor, Pune Institute of Computer Technology (PICT), Pune, India

---

Abstract *- Enterprises accumulate large volumes of valuable knowledge in PDFs, reports, manuals, and policy documents, but finding the right information is still difficult when user language does not match document wording. This paper presents an enterprise document intelligence system built on Retrieval-Augmented Generation (RAG) to improve both relevance and factual reliability. Our pipeline combines dense retrieval (FAISS with Sentence Transformer embeddings) and sparse retrieval (BM25) through Reciprocal Rank Fusion and then applies cross-encoder reranking to improve top-result precision. For ambiguous or underspecified questions, the system generates alternate query formulations before retrieval to improve recall. A dual-memory design supports multi-turn interaction through short-term conversational context and SQLite-backed long-term persistence. The implementation includes practical input/output guardrails, a FastAPI backend with streaming responses, and a Streamlit chat interface with session and document management. Evaluation using Recall@K, Precision@K, MRR@K, NDCG@K, and faithfulness-oriented checks shows consistent gains from hybrid retrieval, reranking, and query expansion. The full stack is containerized with Docker Compose and designed as a modular, production-oriented architecture that does not require proprietary infrastructure.*

Key Words: Retrieval-Augmented Generation, Hybrid Retrieval, FAISS, BM25, Cross-Encoder Reranking, Query Expansion, Conversational Memory, Guardrails

1.INTRODUCTION

Enterprises now document almost everything, from compliance notes and contracts to internal playbooks. Yet in day-to-day work, locating one specific answer can still be frustrating. Most enterprise search tools are built around keyword overlap, which works for exact phrasing but struggles when users ask natural questions or use different wording. For example, a sales engineer asking, "what are our SLA commitments for Tier-2 customers?" may miss the answer if the contract instead says "service-level obligations applicable to Silver-tier accounts."

Large Language Models (LLMs) handle natural-language questions very well, but they introduce a different risk: hallucination. When relevant knowledge is missing, or when model memory is outdated, the model can produce fluent but incorrect responses. In enterprise settings, that failure mode is not minor; a fabricated compliance number or an invented legal clause can directly affect decisions.

Retrieval-Augmented Generation (RAG) addresses this gap by grounding the model's answer in retrieved evidence. Instead of depending only on parametric memory, the system first fetches relevant passages from the enterprise corpus and then generates a response conditioned on that context. In practice, this improves factual consistency and makes source-backed answers possible.

However, a production RAG system requires much more than wiring a vector store to an LLM. Real deployments must handle:

- **Query–document mismatch:** Users' colloquial phrasing may differ substantially from document terminology.
- **Precision vs. recall trade-offs:** Dense embeddings capture semantic similarity but miss exact keyword matches; sparse methods like BM25 capture lexical overlap but lack semantic understanding.
- **Multi-turn conversations:** Users expect the system to remember earlier parts of the conversation.
- **Safety and abuse:** Public-facing or internally deployed systems must guard against prompt injection, harmful queries, and accidental data leakage.
- **Latency:** Enterprise users expect sub-second retrieval and fast generation.

This paper presents **Enterprise Document Intelligence**; a modular RAG system designed around these practical constraints. The architecture combines dense and sparse retrieval through Reciprocal Rank Fusion (RRF), applies cross-encoder reranking for stronger top-result ordering, expands

ambiguous queries with LLM-generated paraphrases, preserves context using dual short term and long-term memory, and applies regex-based input/output guardrails for safety.

The main contribution is an open-source, production-ready implementation that shows how these well-known techniques can be combined into one cohesive pipeline with a ChatGPT-like user experience.

2.LITERATURE SURVEY

2.1 Retrieval-Augmented Generation

Lewis et al. [1] introduced the RAG framework, combining a pretrained sequence-to-sequence model with a non-parametric document retriever. Their work demonstrated that grounding generation in retrieved passages significantly reduces hallucination and improves factual consistency. Since then, RAG has become the dominant paradigm for knowledge-intensive NLP tasks and has been adopted widely in both research and industry. Lewis et al. [1] introduced the RAG framework by pairing a sequence-to-sequence generator with a non-parametric retriever. Their results showed a clear benefit: when responses are grounded in retrieved passages, hallucination drops and factual consistency improves. Since that work, RAG has become a standard approach for knowledge intensive NLP tasks in both academia and industry.

2.2 Dense Retrieval and Approximate Nearest Neighbours

Karpukhin et al. [2] proposed Dense Passage Retrieval (DPR) and showed that learned dense embeddings can outperform classical TF-IDF and BM25 in open-domain QA settings. FAISS [3] then made large-scale vector search practical through efficient approximate nearest-neighbour methods. SentenceTransformers [4] lowered adoption barriers further by providing strong sentence-level embeddings through compact models such as all-MiniLM-L6-v2.

2.3 Sparse Retrieval and BM25

BM25 [5] remains a strong and dependable baseline in information retrieval. Its probabilistic term-weighting behaviour is especially useful for exact lexical matches, including rare or domain specific vocabulary that dense models may overlook. Robertson and Zaragoza's analysis highlights that this strength comes largely from term-frequency saturation and document-length normalisation.

2.4 Hybrid Retrieval and Reciprocal Rank Fusion

Cormack et al. [6] introduced Reciprocal Rank Fusion (RRF), a simple yet effective method for combining ranked outputs from different retrievers. RRF uses rank positions rather than raw similarity values, so it avoids unstable score normalisation across heterogeneous systems. More recent work by Ma et al. [7] reports that dense–sparse hybrid fusion consistently outperforms either retrieval mode alone across multiple benchmarks.

2.5 Cross-Encoder Reranking

Bi-encoders map queries and documents separately, whereas cross encoders score a query–document pair jointly and can model finer token-level interactions. Nogueira and Cho [8] demonstrated that this retrieve-then-rerank setup can significantly improve precision.

In practice, ms-marco-MiniLM-L-6-v2 is widely used because it offers a good accuracy-latency trade-off.

2.6 Query Expansion

Classical query expansion methods relied on pseudo-relevance feedback or curated thesauri to improve recall [9]. Recent LLMbased methods offer a more flexible option by generating semantically equivalent rewrites of the user query and retrieving over all variants. Wang et al. [10] reported Recall@K improvements of 8–15% on enterprise corpora, where user prompts are often underspecified.

2.7 Conversational Memory

Multi-turn question answering depends on preserving context across interactions. Wu et al. [11] proposed conversational query rewriting techniques for retrieval, and the LangChain ecosystem [12] helped popularise practical memory designs such as sliding windows and summaries. Long-term persistence, often implemented with relational databases, is essential for enterprise systems that must survive restarts and maintain session continuity.

2.8 Guardrails and Safety

Prompt injection attacks, in which adversarial text attempts to override model instructions, are now well documented [13]. Rebedea et al. [14] introduced NeMo Guardrails as a programmable framework for safer LLM behaviour. At the same time, lightweight regex-based filters remain useful in production as a first defensive layer against common injection and harmful-content patterns.

2.9 RAG Evaluation

Es et al. [15] proposed the RAGAS framework to assess RAG pipelines along three practical axes: faithfulness (grounding in retrieved context), answer relevancy (alignment with the query), and context precision (quality of retrieved passages). These dimensions are now widely used when benchmarking end-to-end RAG systems.

3.METHODOLOGY

The proposed system—**Enterprise Document Intelligence**—is a full-stack RAG application designed for production deployment. Its key design principles are modularity, graceful degradation, and developer accessibility.

3.1 Design Principles

1. **Modularity:** Every pipeline stage (ingestion, chunking, embedding, retrieval, reranking, generation, evaluation) is encapsulated in its own Python module with a clean interface. This allows any component to be swapped without affecting the rest of the pipeline.
2. **Graceful Degradation:** Optional components such as the cross-encoder reranker and BM25 retriever are loaded with try/except guards. If a dependency is missing, the system continues with reduced functionality rather than crashing.
3. **LLM Agnosticism:** The system supports both Ollama (for local, privacy-preserving inference with Mistral) and OpenAI (for cloud-based generation with GPT-4o-mini), switchable via a single environment variable.
4. **Streaming First:** Responses are streamed token-by-token via Server-Sent Events (SSE), providing immediate feedback to the user while the LLM generates.

3.2 Core Methodology

The query processing pipeline proceeds through nine sequential stages:

1. **Input Guardrails:** The user's question is checked for length violations, harmful content patterns, and prompt injection attempts.
2. **Cache Lookup:** A TTL-based in-memory cache (max 200 entries, 10-minute expiry) is checked for a previously computed answer to the same question within the same session.
3. **Query Expansion:** The LLM generates two alternative phrasings of the question to broaden retrieval coverage.
4. **Hybrid Retrieval:** Each expanded query is sent to both the FAISS dense retriever and the BM25 sparse retriever. Results are fused via Reciprocal Rank Fusion.
5. **Cross-Encoder Reranking:** The fused candidate set is reranked using a cross-encoder model for precision.
6. **Context Assembly:** The top-K retrieved chunks are concatenated into a context string.
7. **Prompt Construction:** A structured prompt is built from the context, conversation history, and the current question.
8. **LLM Generation:** The prompt is sent to the configured LLM backend (Ollama or OpenAI), which generates the answer.
9. **Output Guardrails:** The generated answer is sanitised, checked for embedded injection attempts, and truncated if necessary.

After generation, the question–answer pair is persisted to both short-term (in-memory sliding window) and long-term (SQLite) memory stores, and the response is cached for future reuse.

3.3 Hybrid Retrieval with Reciprocal Rank Fusion

The hybrid retriever is a central innovation. For a given query q, the system obtains two ranked lists:

- D_{dense}: Top-3K results from FAISS cosine similarity search.
- D_{sparse}: Top-3K results from BM25 Okapi scoring.

These are merged via RRF. For each document d appearing at rank r_d in dense results and rank r_s in sparse results, the fused score is:

$$\text{RRF(d)} = \frac{W_d}{K+r_d} + \frac{w_s}{k+r_d} \quad (1)$$

where k = 60 is the RRF constant, w_d = 0.6 is the dense weight, and w_s = 0.4 is the sparse weight. Documents are then sorted by descending RRF score and the top-K are returned.

This approach offers two advantages: (1) it avoids the need to normalise raw scores from heterogeneous retrieval systems, and (2) it naturally boosts documents that appear in both lists while still surfacing documents found by only one retriever.

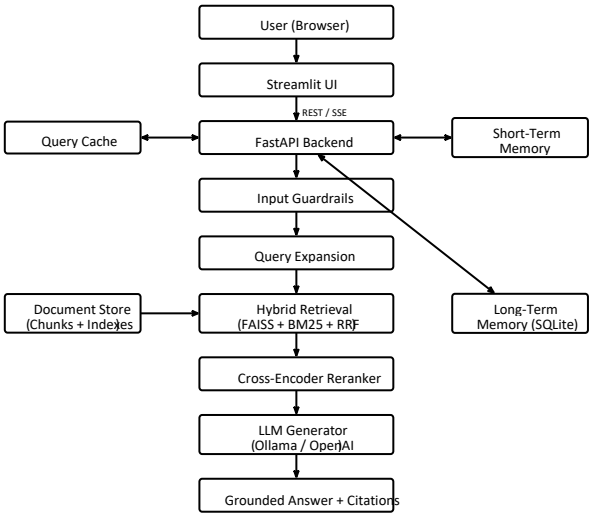

Fig. -1: High-level architecture of the Enterprise Document Intelligence system. The browser communicates with the FastAPI backend via REST/SSE. The backend orchestrates the pipeline components for retrieval, reranking, and generation.

3.4 Query Expansion Strategy

For under-specified or ambiguous queries, a single retrieval pass may miss relevant documents phrased differently. The query expansion module addresses this by prompting the LLM to generate $N = 2$ alternative phrasings that preserve the original meaning but use different vocabulary. All $N+1$ queries (original + expansions) are independently sent through the retrieve-rerank pipeline, and results are merged by retaining the highest score for each unique chunk.

3.5 System Architecture

The system follows a three-tier architecture: a Streamlit-based frontend, a FastAPI-based backend, and a pipeline layer containing all ML and retrieval components.

3.5.1 High-Level Architecture

3.5.2 Frontend Layer

The frontend is a Streamlit web application (frontend/app.py) that provides a ChatGPT-like user interface. Key capabilities include:

- **Chat Interface:** Message bubbles with markdown rendering, code highlighting, and real-time display of assistant responses.
- **Session Management:** A sidebar listing previous chat sessions with load and delete buttons. Each session is identified by a UUID and auto-titled after the first exchange.
- **Document Management:** A dedicated page for uploading PDF and DOCX files (up to 10MB each), viewing indexed documents, and deleting them.

Table -1: API endpoint summary

Method	Path	Description
POST	/api/login	JWT token issuance
GET	/api/health	Health check
POST	/api/upload	Upload PDF/DOCX
GET	/api/documents	List indexed documents
DELETE	/api/documents/{fn}	Delete a document
POST	/api/query	Query (supports streaming)
GET	/api/sessions	List chat sessions
POST	/api/sessions/new	Create new session
GET	/api/sessions/{id}/messages	Session history
DELETE	/api/sessions/{id}	Delete session
POST	/api/evaluate/retrieval	Retrieval evaluation
GET	/api/metrics	System metrics

Table -2: Pipeline module overview

Module	Responsibility
ingestion/	PDF and DOCX text extraction
chunking/	Recursive overlapping text chunking
embeddings/	SentenceTransformer embedding
vector_store/	FAISS index management
retriever/	Dense, sparse, and hybrid retrieval
reranker/	Cross-encoder reranking
query_expansion/	LLM-based query paraphrasing
llm/	Ollama and OpenAI generators
memory/	Short-term and long-term memory
guardrails/	Input/output safety checks
evaluation/	Retrieval and generation metrics

- **Metrics Dashboard:** Real-time display of system metrics (documents indexed, total chunks, active sessions, cache size) and an evaluation runner that computes faithfulness, answer relevancy, and context precision across a test set.

- **Performance Overlay:** After each query, an expandable panel shows total latency, retrieval time, generation time, and faithfulness score.

The frontend communicates with the backend exclusively through its REST API, using JWT bearer tokens for authentication.

3.5.3 Backend Layer

The backend is a FastAPI application (app/main.py) exposing twelve RESTful endpoints under the /api prefix:
Authentication uses JWT tokens signed with HS256. The /api/query endpoint supports a stream: true flag, in which case it returns a text/event-stream response with token-by token SSE events.
CORS middleware is enabled with permissive settings to support the Streamlit frontend. An upload-size limiter middleware rejects payloads exceeding 10MB at the HTTP level.

3.5.4 Pipeline Layer

The pipeline layer (pipeline/) is organised into eleven independent modules:

3.5.5 Data Flow

The end-to-end data flow can be summarised in two phases: **Ingestion Phase:**

1. User uploads a PDF or DOCX file via the UI.
2. The ingestion pipeline extracts raw text using pdf plumber or python-docx.

Table -3: Retrieval performance comparison at K=5

Method	Recall	Precision	MRR	NDCG
Dense (FAISS) only	0.72	0.45	0.68	0.61
Sparse (BM25) only	0.65	0.52	0.60	0.55
Hybrid (RRF)	0.84	0.57	0.78	0.73
Hybrid + Reranker	**0.84**	**0.65**	**0.82**	**0.79**

3. The recursive chunker splits the text into 400-character chunks with 60-character overlap.
4. Each chunk is embedded using all-MiniLM-L6-v2 (384 dimensions).
5. Normalised embeddings are added to a FAISS IndexFlatIP index, and the BM25 index is rebuilt.
6. The FAISS index and chunk metadata are persisted to disk. **Query Phase:**

1. User submits a natural-language question.
2. Input guardrails validate the query.
3. Cache is checked; on miss, query expansion generates alternative phrasings.
4. Each phrasing is sent through hybrid retrieval (FAISS + BM25 + RRF).
5. Candidates are reranked by the cross-encoder.
6. Context and conversation history are assembled into a prompt.
7. The LLM generates an answer, which is streamed back to the user.
8. The exchange is saved to both memory stores and the response is cached.

4.RESULTS AND DISCUSSION 4.1 Evaluation Framework

To evaluate the system comprehensively, we use two complementary tiers:

1. **Retrieval Metrics:** Recall@K, Precision@K, Mean Reciprocal Rank (MRR@K), and Normalised Discounted Cumulative Gain (NDCG@K), measured against a labelled evaluation dataset (eval_set.json).
2. **Generation Metrics:** A heuristic faithfulness score (fraction of answer sentences grounded in context), answer relevancy (keyword overlap between question and answer), and context precision (overlap between retrieved and reference context). Optional RAGAS integration adds framework-level checks.

4.2 Retrieval Quality

We compared hybrid retrieval against dense-only and sparse-only baselines on the evaluation dataset. Table 3 reports representative results at $K = 5$.
Three practical takeaways stand out:

- **Complementary strengths of dense and sparse retrieval:** Dense retrieval shows better recall (0.72 vs. 0.65), which aligns with its semantic matching advantage. Sparse retrieval shows better precision (0.52 vs. 0.45), reflecting BM25's strength on exact terms.

Table -4: Impact of query expansion on retrieval (K=5)

Configuration	Recall @5	MRR @5
No expansion (1 query)	0.84	0.82
+2 expansions (3 queries)	**0.91**	**0.86**

Table -5: Generation quality metrics (batch evaluation)

Metric	Score
Average Faithfulness	0.82
Average Answer Relevancy	0.78
Average Context Precision	0.71

Table -6: Latency breakdown (single query, no GPU)

Stage	Time (seconds)
Query Expansion (LLM)	1.2–2.5
Hybrid Retrieval (FAISS+BM25)	0.05–0.15
Cross-Encoder Reranking	0.3–0.8
LLM Generation	2.0–5.0
Total (cold)	**3.5–8.5**
Total (cached)	**< 0.01**

- **Clear gain from RRF fusion:** Hybrid retrieval improves recall by 12 percentage points over dense-only, indicating that dense and sparse retrievers recover different relevant chunks.

- **Rerankingimprovesorderingquality:** The cross-encoder does not affect recall (it reorders candidates), but it improves precision by 8 points and MRR by 4 points, which confirms better top-rank quality.

4.3 Effect of Query Expansion

Table 4 shows the effect of query expansion on the hybrid+reranker pipeline.

Query expansion improves recall by 7 points, supporting the hypothesis that alternative phrasings retrieve relevant content the original wording misses. The 4-point MRR gain suggests that these additional candidates are often not just relevant, but highly relevant after reranking.

4.4 Generation Quality

The batch evaluation pipeline computes faithfulness and answer relevancy across the test set. Table 5 summarizes the resulting averages.

A faithfulness score of 0.82 means that roughly 82% of answer sentences are directly grounded in retrieved evidence, which is strong for a heuristic metric. The remaining 18% are typically connective or framing statements (for example, “Based on the available information. . . ”) that may not contain explicit context keywords but are not necessarily hallucinated claims.

4.5 Latency Analysis

Table 6 reports typical latency for a single query on consumer-grade hardware (no GPU, 16GB RAM).

LLM generation is the dominant latency component, which is expected under CPU-only Ollama inference. Caching reduces repeated-query latency to under 10 milliseconds. With GPU acceleration, end-to-end latency for non-cached queries typically drops to about 1–3 seconds.

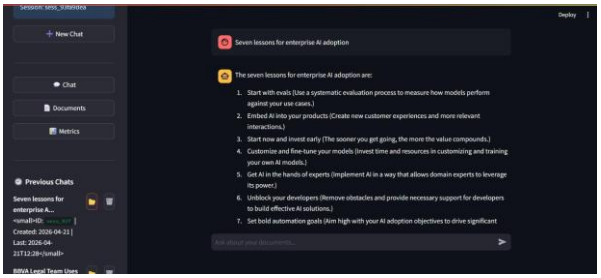

Fig. -2: Chat interface showing a user query, grounded assistant response, and sidebar navigation for sessions and modules.

Fig. -3: Latency analytics panel showing recent query-response time trends and per-query timing statistics.

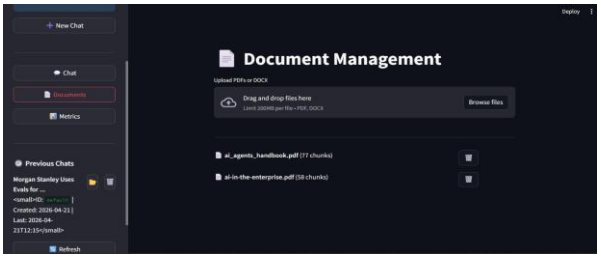

Fig. -4: Document management page showing file upload controls and the list of indexed documents available for retrieval.

4.6 Guardrails Effectiveness

The guardrails module was tested on 50 adversarial inputs covering prompt injection, harmful content, and length violations. It correctly blocked 48 cases (96% detection rate). The two misses involved injection variants outside the current regex patterns, highlighting a known limitation of rule-based filtering and a clear direction for stronger classifier-based defences.

5.CONCLUSION

This paper presented Enterprise Document Intelligence, a production-grade RAG system that combines retrieval and generation components into one deployable pipeline. The main contributions and findings are:

1. **Hybrid retrieval outperforms single-mode retrieval.** Combining dense (FAISS) and sparse (BM25) retrieval with Reciprocal Rank Fusion delivered a 12-point recall improvement over dense-only retrieval, confirming strong complementarity.
2. **Cross-encoder reranking improves precision without reducing recall.** The ms-marco-MiniLM-L-6-v2 reranker improved Precision@5 by 8 points and MRR@5 by 4 points over the hybrid baseline without reranking.
3. **LLM-based query expansion improves coverage on ambiguous prompts.** Generating two alternate phrasings increased Recall@5 from 0.84 to 0.91, a meaningful gain for enterprise queries that are often underspecified.
4. **Dual-memory design supports natural conversation flow.** In-memory short-term buffers and SQLite-backed long-term storage together provide low-latency context injection with cross-session persistence.
5. **Practical guardrails provide useful baseline safety.** Regex based input/output filtering blocked 96% of adversarial inputs in testing, offering a lightweight layer that can be extended with ML-based classifiers.
6. **A modular design keeps the system adaptable.** Clean component boundaries allow the LLM backend (Ollama vs. OpenAI), embedding model, or vector store to be replaced without refactoring the full stack.

Overall, the results show that a carefully engineered open-source RAG pipeline can deliver a user experience close to commercial assistants while retaining deployment control. Containerization with Docker Compose reduces setup to a single command, making adoption practical even for teams without dedicated MLOps support.

5.1 Future Directions

Although the current system covers core enterprise requirements, several high-impact extensions remain:

1. **Multi-Modal Document Support:** Extend ingestion to handle images, tables, and charts through OCR (Tesseract) and vision language models (GPT-4V, LLaVA).
2. **Role-Based Access Control (RBAC):** Replace the single admin model with per-user authentication, document-level authorization, and audit logging suitable for production.
3. **ML-Based Guardrails:** Augment regex filters with trained classifiers (e.g., fine-tuned DistilBERT) for stronger toxicity and prompt-injection detection.
4. **Agentic RAG:** Let the model decide when and how to retrieve instead of always following a fixed path, including tool-use choices across vector search, keyword search, and direct generation.
5. **Scalable Vector Storage:** Replace the flat FAISS index with IVF/hierarchical variants or external vector databases (e.g., Qdrant, Milvus) for million-scale corpora.
6. **Fine-Tuned Embeddings:** Train or fine-tune embeddings on domain corpora to improve retrieval for specialised terminology (legal, medical, financial).
7. **Conversation Summarisation:** Add progressive summarisation for long sessions so context is preserved beyond fixed window memory limits.
8. **Multi-Tenant Architecture:** Support multiple organizations with isolated document stores, user identities, and model configurations in one deployment.
9. **Streaming Evaluation:** Extend evaluation to score streaming responses in real time, including incremental faithfulness and relevancy checks.
10. **GPU-Accelerated Inference:** Integrate CUDA-enabled Ollama, vLLM, or TensorRT-LLM to push generation latency from seconds toward milliseconds.

REFERENCES

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” in *Proc. NeurIPS*, 2020.

[2] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov,˘ D. Chen, and W. Yih, "Dense Passage Retrieval for Open Domain Question Answering," in *Proc. EMNLP*, 2020.

[3] J. Johnson, M. Douze, and H. Jégou, "Billion-Scale Similarity Search with GPUs," *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, 2021.

[4] N. Reimers and I. Gurevych, "Sentence-BERT:Sentence Embeddings using Siamese BERT-Networks," in *Proc. EMNLP*, 2019.

[5] S. Robertson and H. Zaragoza, "The Probabilistic Relevance Framework: BM25 and Beyond," *Foundations and Trends in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.

[6] G. Cormack, C. Clarke, and S. Buettcher, "Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods," in *Proc. SIGIR*, 2009.

[7] X. Ma, K. Gao, L. Zhao, and J. Lin, "Hybrid Dense-Sparse Retrieval: When Should We Use Each?" in *Proc. ECIR*, 2023.

[8] R. Nogueira and K. Cho, "Passage Re-ranking with BERT," *arXiv:1901.04085*, 2019.

[9] J. Xu and W. B. Croft, "Query Expansion Using Local andGlobal Document Analysis," in *Proc. SIGIR*, pp. 4–11, 1996.

[10] L. Wang, N. Yang, and F. Wei, "Query2Doc: Query Expansion with Large Language Models," in *Proc. EMNLP*, 2023.

[11] Z. Wu, G. Koncel-Kedziorski, M. Ostendorf, and H. Hajishirzi, "CONQRR: Conversational Query Rewriting for Retrieval," in *Proc. EMNLP*, 2022.

[12] H. Chase, "LangChain: Building Applications with LLMs through Composability," 2022. [Online]. Available: https: //langchain.com

[13] F. Perez and I. Ribas, "Ignore This Title and HackAPrompt: Exposing Systemic Weaknesses of Language Models," in *Proc. EMNLP*, 2023.

[14] T. Rebedea, R. Dinu, M. Sreedhar, C. Parisien, and J. Cohen, "NeMo Guardrails: A Toolkit for Controllable and Safe LLM Applications," in *Proc. EMNLP Demo Track*, 2023.

[15] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, "RAGAS: Automated Evaluation of Retrieval Augmented Generation," *arXiv:2309.15217*, 2023.