FoundryRAG: Structure-Aware Hybrid Retrieval with Adaptive Context Reconstruction for Local-Cloud Document Question Answering

Implemented with Microsoft Foundry Local, Azure AI Search, and Azure OpenAI

Mahmut Esat Kolay
AI Intern
Microsoft Türkiye
AI Innovator Summer Internship
July 2026

Architecture • Retrieval Engineering • Local/Cloud Portability • Evaluation

Abstract

This report presents FoundryRAG, a structure-aware hybrid retrieval system for local-cloud technical document question answering, developed during the Microsoft Türkiye AI Innovator Summer Internship. The implementation supports offline execution through Microsoft Foundry Local and an operational cloud deployment using Azure AI Search, Azure OpenAI, and Azure Blob Storage. Its central contribution is an architecture that combines hierarchical document parsing, hybrid sparse/dense retrieval, reciprocal rank fusion, cross-encoder re-ranking, retrieval grading, context compression, parent-section reconstruction, adaptive context budgeting, and one bounded follow-up retrieval hop. The report focuses on architecture, data flow, engineering decisions, latency, and retrieval evaluation against a naive dense baseline.

Index Terms—retrieval-augmented generation, hybrid retrieval, Microsoft Foundry Local, Azure AI Search, cross-encoder re-ranking, document intelligence, explainability.

Document Control

Item	Value
Author	Mahmut Esat Kolay
Role	AI Intern
Organization	Microsoft Türkiye
Internship	AI Innovator Summer Internship
System status	Local and cloud modes completed
Document version	1.0

Contents

1. Introduction

FoundryRAG was designed to move beyond the common tutorial pattern of fixed-size chunking, one embedding model, and raw top-k similarity retrieval. Industrial documentation frequently includes tables, warnings, section hierarchies, equipment identifiers, and maintenance dependencies that lose meaning when flattened into arbitrary windows. The system therefore treats document structure as a first-class data model and retrieval as a staged quality-control process.

The implementation was completed as an engineering-focused internship project. It contains a FastAPI application, modular format parsers, a shared KnowledgeNode representation, hybrid retrieval, re-ranking, grading, compression, source traceability, telemetry, local inference, and an operational Azure cloud mode. The primary goal is not to claim a new foundation model, but to demonstrate how a robust RAG pipeline can be assembled, measured, explained, and migrated across deployment environments.

1.1 Main Contributions

- A shared hierarchical representation for PDF, DOCX, Markdown, XLSX, and CSV content.
- Hybrid BM25 and dense retrieval fused with Reciprocal Rank Fusion and cross-track deduplication.
- Cross-encoder re-ranking followed by relevance grading and context compression.
- Full parent-section reconstruction to restore context around a matched chunk.
- Score-triggered, bounded follow-up retrieval for weak initial evidence.
- Local execution through Foundry Local and completed Azure-based retrieval and generation.
- Chunk-level explainability, source provenance, latency telemetry, and persistent query logging.

2. System Objectives and Design Principles

The system was guided by five engineering principles: preserve structure before chunking, combine lexical and semantic evidence, bound adaptive behavior, fail honestly, and separate application logic from infrastructure-specific services.

Objective	Implementation response
Offline-first operation	Foundry Local generation, SentenceTransformer embeddings, SQLite persistence, and local cached retrieval indexes.
High retrieval coverage	Query expansion, BM25, dense vectors, RRF fusion, re-ranking, and one bounded extra hop.
Context integrity	KnowledgeNode hierarchy, atomic structured nodes, parent-section reconstruction, and adaptive context budget.
Transparency	Source citations, selected-chunk matrix, re-rank scores, stage latency, and query logs.
Cloud portability	Backend interfaces for Azure AI Search, Azure OpenAI, and Blob-backed document synchronization.

3. High-Level Architecture

The platform is organized into the client, FastAPI backend, document-processing pipeline, RAG pipeline, knowledge storage, and model backends. The client communicates exclusively with the API. Ingestion writes structured nodes, chunks, and graph data to the persistence layer. Retrieval reads indexed chunks and returns ranked evidence to generation. Local and cloud model paths expose the same application-level contracts.

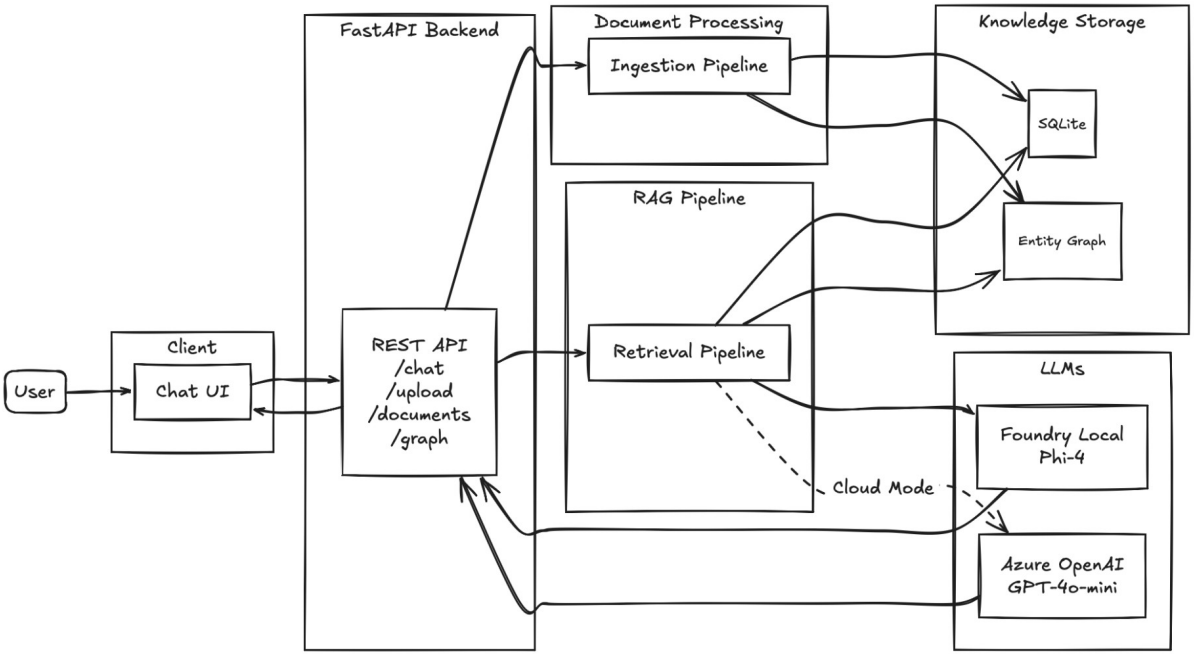

Fig. 1. Overall system architecture showing client, FastAPI, ingestion, retrieval, storage, and model backends.

4. Ingestion Architecture

Ingestion normalizes heterogeneous formats into a common KnowledgeNode tree. This is a deliberate inversion of chunk-first design: chunks are derived views over a structured source model. Heading, paragraph, table, figure, warning, note, and code nodes retain page, section, and parent identifiers. Structured nodes such as tables and warnings are indexed atomically, while oversized tabular content can be split by row range.

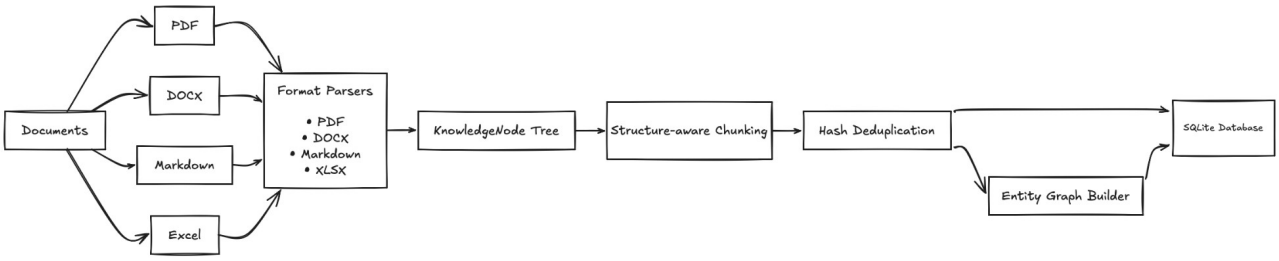

Fig. 2. Structure-aware document ingestion pipeline.

4.1 Incremental Indexing and Entity Extraction

Content hashes prevent redundant reprocessing. Unchanged documents are skipped; changed documents are removed and re-indexed to avoid stale records. The entity graph builder extracts technical terms per section and constructs co-occurrence edges in code. This graph is persisted alongside the document hierarchy and can support both visualization and future graph-assisted retrieval.

Stage	Input	Output	Control
Parse	Source file	KnowledgeNode tree	Shared parser interface and node taxonomy
Chunk	Node tree	Retrieval chunks	Heading boundaries; atomic structured nodes
Hash check	Source bytes	Skip/re-index decision	No duplicate unchanged documents
Embed	Chunk text	Dense vectors	Embedding model compatibility validation
Entity extraction	Document sections	Entities and edges	Bounded output and repetition protection

5. Query Processing and Hybrid Retrieval

Every user request first passes through a query-processing stage. The original query is retained and an LLM-based rewriter produces domain-agnostic variants, up to three tracks in total. Each track is embedded with the active local or cloud embedding backend.

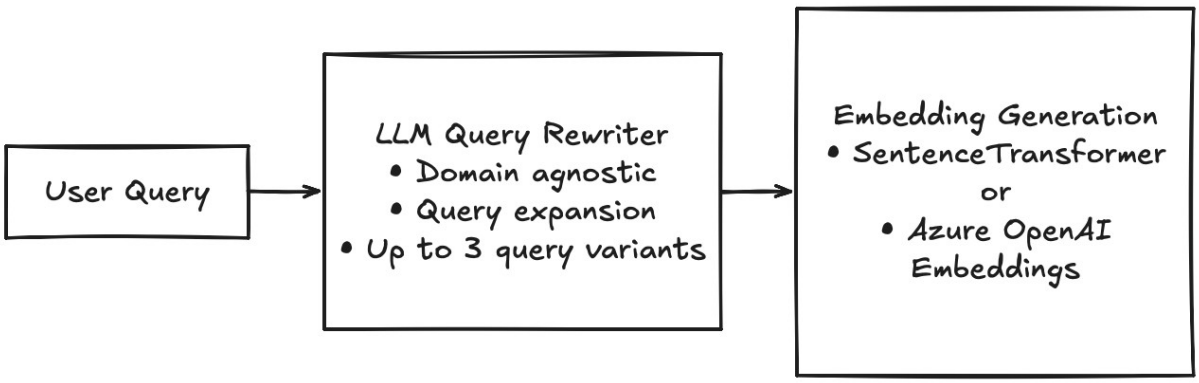

Fig. 3. Query rewriting and embedding generation.

5.1 Sparse and Dense Candidate Retrieval

BM25 is used to recover exact terminology, identifiers, and lexical matches. Dense cosine similarity captures semantic paraphrases. Reciprocal Rank Fusion combines rank positions without requiring raw score calibration. Candidates from all tracks are then deduplicated by chunk identifier.

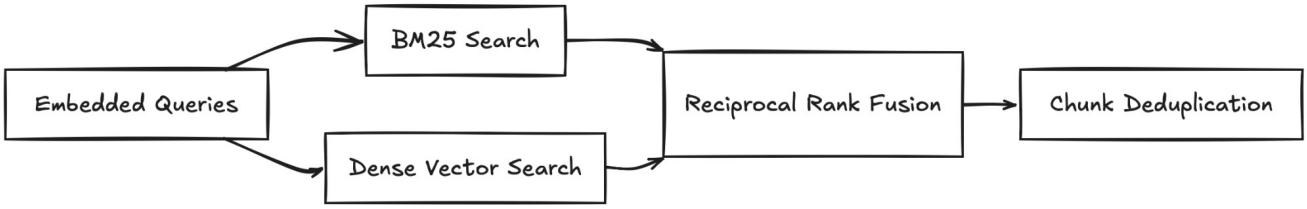

Fig. 4. Multi-track hybrid retrieval and Reciprocal Rank Fusion.

6. Context Optimization and Adaptive Retrieval

Hybrid retrieval increases candidate coverage, but raw candidates are not directly injected into the generation prompt. A cross-encoder jointly scores the query and passage, after which the grader removes near-duplicates and applies score and keyword relevance filters. Context compression prunes ordinary prose to relevant windows while preserving tables, figures, notes, and warnings. Parent-section reconstruction then retrieves sibling nodes under the same parent to restore the surrounding technical context.

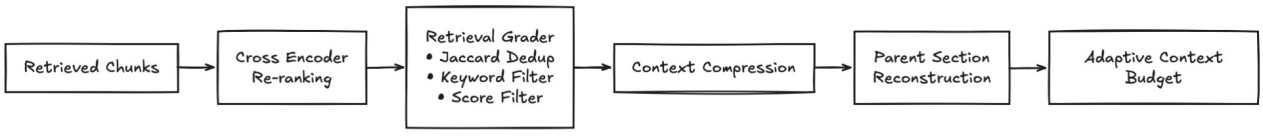

Fig. 5. Re-ranking, retrieval grading, context compression, parent reconstruction, and adaptive context budgeting.

6.1 Adaptive Context Budget

The number of included chunks is adjusted according to the top retrieval score. High-confidence evidence uses a smaller context budget, while weaker evidence retains more candidate material. This reduces prompt noise without enforcing a fixed number of chunks for every question.

6.2 Bounded Follow-up Retrieval

If the score remains below the configured confidence threshold, the request is decomposed into sub-queries and exactly one additional retrieval round is executed. The results are merged into the final context. The strict one-hop limit protects predictable latency and avoids open-ended agentic behavior.

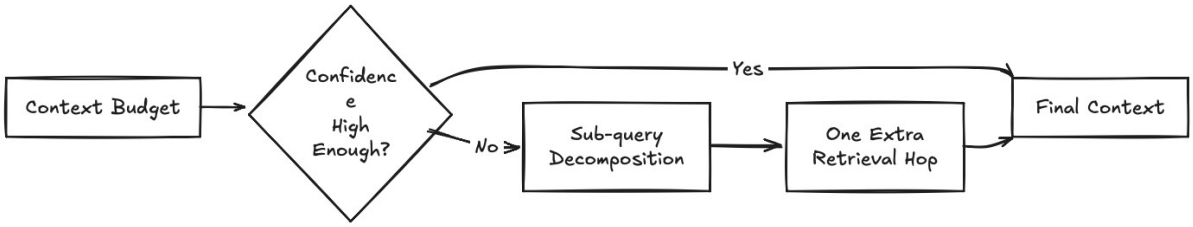

Fig. 6. Score-triggered bounded follow-up retrieval.

7. Answer Generation and Reliability

The packaged context includes source file, page, node type, and section information for each evidence item. The generation prompt requires synthesis from the supplied evidence rather than chunk-by-chunk narration. Local mode uses Foundry Local with Phi-4-mini; cloud mode uses Azure OpenAI through the same generation interface.

A repetition-loop detector inspects generated output for degenerate patterns. When a loop or generation error is detected, the system returns an honest failure message while preserving the retrieved references for inspection. Successful requests return the final answer, reference list, telemetry, chunk matrix, and a persistent query-log event.

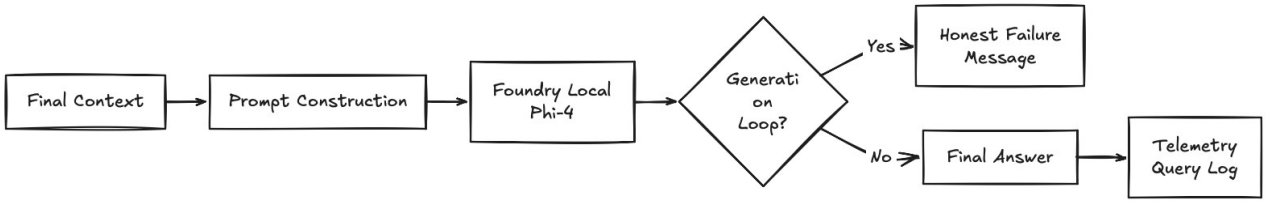

Fig. 7. Answer generation, repetition detection, failure handling, and telemetry logging.

7.1 Complete Query Pipeline

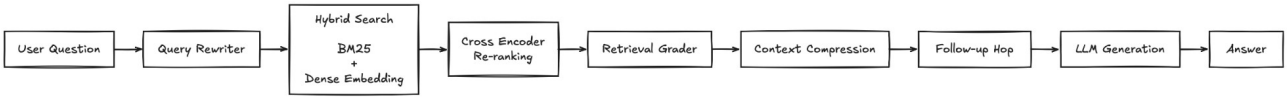

Fig. 8. End-to-end retrieval-augmented generation pipeline.

8. Local and Cloud Deployment Architecture

FoundryRAG implements two operational modes behind the same FastAPI application. Local mode uses all-MiniLM-L6-v2 embeddings, local BM25 and cosine retrieval, SQLite, and Foundry Local generation. Cloud mode uses Azure Blob Storage for document synchronization, Azure AI Search for lexical/vector retrieval, and Azure OpenAI for generation. The application-level re-ranking, grading, compression, parent expansion, adaptive budget, and telemetry logic remain consistent across modes.

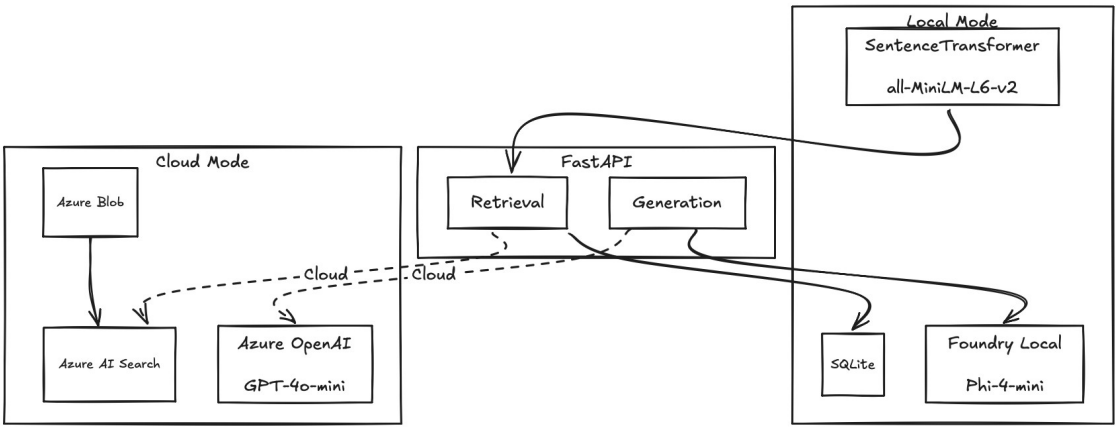

Fig. 9. Completed local and Azure cloud deployment architecture.

8.1 Backend Compatibility

The cloud retrieval adapter returns the same logical fields as local retrieval: chunk text, source file, page, node type, parent identifier, and score. This compatibility prevents cloud migration from becoming a rewrite of the RAG pipeline. It also makes local and cloud evaluation comparable at the orchestration level.

9. Data, API, and Observability Design

9.1 Persistence Model

Data object	Purpose
documents	Document registry, source metadata, and content hashes.
knowledge_nodes	Hierarchical representation of parsed document structure.
document_chunks	Retrieval units, metadata, and dense vectors.
entities / entity_edges	Extracted technical concepts and co-occurrence relationships.
query_log	Persistent query outcomes, selected evidence, and timing data.

9.2 API Surface

Endpoint	Responsibility
POST /chat	Execute the full RAG pipeline and return answer, references, reasoning metadata, and telemetry.
POST /upload	Upload and ingest a supported document.
GET /documents	List indexed documents and chunk counts.
GET /graph	Return entity-graph nodes and edges.
GET /source/{file}	Open source evidence with page targeting.
GET /health	Report application and backend health.

9.3 Explainability and Telemetry

The interface exposes the number and type of injected chunks, the source document, evidence excerpt, re-rank score, expanded query tracks, retrieval-hop count, and per-stage latency. This instrumentation was essential for discovering that short query-rewriting calls were consuming an unnecessarily large generation budget.

10. Performance Analysis

Table I reports representative local latency on Apple Silicon M4. Values depend on corpus size, model state, and hardware load; they characterize the pipeline rather than define a service-level objective.

Stage	Latency	Notes
Query expansion	~4.0 s	LLM rewrite capped at max_tokens=120; approximately 27 s before optimization.
Vector embedding	~83 ms	Local SentenceTransformer on CPU.
Hybrid sparse/dense retrieval	~64 ms	Cached BM25 and vectorized cosine similarity.
Cross-encoder re-rank	~807 ms	bge-reranker-base over the top six candidates.
Grade + compress	~5 ms	Jaccard deduplication and parent-section reconstruction.

Stage	Latency	Notes
Token generation	~6.5 s	Full synthesis with max_tokens=1000 to avoid truncation.

10.1 Latency Interpretation

Generation and query expansion dominate end-to-end response time because both invoke a local language model. The cross-encoder is the principal retrieval-side cost. Embedding, hybrid retrieval, grading, and reconstruction remain comparatively small. The observed profile justifies caching indexes, capping short structured generations, and limiting follow-up retrieval to one hop.

11. Evaluation Methodology and Results

The retrieval evaluation compares the advanced pipeline with a naive dense baseline on the same labeled evaluation set and a Top-5 retrieval cutoff. Precision@5 measures context cleanliness, Recall@5 measures evidence coverage, and Mean Reciprocal Rank measures how early the first relevant result appears.

11.1 Retrieval Results

Pipeline	Precision@5	Recall@5	MRR
Naive Retrieval	0.857	0.857	0.857
Advanced Retrieval Pipeline	0.893	0.982	0.905

11.2 Improvement over Baseline

Metric	Absolute change	Relative improvement
Precision@5	+0.036	+4.2%
Recall@5	+0.125	+14.6%
MRR	+0.048	+5.6%

The advanced pipeline improves all three retrieval metrics. The largest gain is in recall, indicating that the multi-stage pipeline misses fewer relevant evidence items within the first five results. The MRR increase shows that correct evidence also tends to move earlier in the ranking. These results support the combined architecture, although component-level attribution requires an ablation study.

11.3 Recommended Ablation

Configuration	Purpose
Dense-only baseline	Establish semantic-retrieval reference performance.
BM25 + dense + RRF	Measure the contribution of hybrid retrieval.
Hybrid + cross-encoder	Measure ranking improvement.
Full pipeline	Measure grading, compression, parent reconstruction, and bounded hop together.

12. Engineering Decisions

Problem observed	Decision	Engineering effect
Flat chunks lost section context	KnowledgeNode tree and parent reconstruction	Preserves document structure and restores sibling context.
Dense retrieval missed exact identifiers	BM25 + dense retrieval with RRF	Combines lexical precision and semantic recall.
Hybrid index rebuild was costly	Cache index until corpus changes	Keeps retrieval in the millisecond range.
Small-model output repeated prompt fragments	Loop detection and Phi-4-mini	Returns explicit failure instead of unusable text.
Static synonym expansion was domain-specific	LLM-based query rewriting	Supports document-independent terminology.
One-shot retrieval missed multi-part questions	One bounded follow-up hop	Improves coverage without unbounded agent latency.
Short LLM calls used excessive tokens	Per-call token budget	Reduced rewriting latency from ~27 s to ~4 s.
Local/cloud vector dimensions can differ	Embedding compatibility guard	Fails early with an actionable configuration error.

13. Security and Operational Considerations

- Local mode minimizes routine transmission of proprietary engineering documents.
- Cloud mode should use Microsoft Entra ID or managed identities, role-based access control, and private network boundaries.
- Upload endpoints require MIME validation, size limits, filename sanitization, and path-traversal protection.
- Prompt construction treats retrieved content as untrusted data and should separate system instructions from evidence.
- Source documents, SQLite files, search indexes, and query logs require explicit retention and deletion policies.
- Azure OpenAI usage should be protected by quotas, budget alerts, and endpoint rate limits.

14. Limitations and Future Work

The project is functionally complete in local and cloud modes, but further work is needed to strengthen scientific evidence and production hardening.

- Expand the evaluation set across more document types, domains, and adversarial questions.
- Add component-level ablation, confidence intervals, and statistical significance testing.
- Integrate the entity graph as an explicit retrieval signal.
- Add automated unit, integration, and end-to-end regression tests.
- Add CI-based benchmark generation and model/backend comparison reports.
- Implement authentication, tenant isolation, and production-grade operational monitoring.
- Evaluate vision captioning for figures and diagrams in technical manuals.

15. Conclusion

FoundryRAG demonstrates a complete, modular RAG system in which retrieval quality, document structure, reliability, and observability are treated as core engineering concerns. The architecture combines structure-aware ingestion, hybrid sparse/dense retrieval, cross-encoder re-ranking, grading, compression, parent reconstruction, adaptive budgeting, and a bounded follow-up hop. It runs both locally through Microsoft Foundry Local and in Azure through Azure AI Search, Azure OpenAI, and Blob Storage.

The measured retrieval results show gains over the naive dense baseline in precision, recall, and ranking quality. The latency trace identifies local generation as the dominant cost and validates the optimization of short LLM calls. Overall, the project provides a strong engineering foundation for industrial document question answering, explainable evidence retrieval, and controlled local-to-cloud deployment.

References

[1] P. Lewis et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” in Advances in Neural Information Processing Systems, vol. 33, 2020, pp. 9459–9474.

[2] 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.

[3] G. V. Cormack, C. L. A. Clarke, and S. Büttcher, “Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods,” in Proc. 32nd ACM SIGIR, 2009, pp. 758–759.

[4] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” in Proc. EMNLP-IJCNLP, 2019, pp. 3982–3992.

[5] R. Nogueira and K. Cho, “Passage Re-ranking with BERT,” arXiv:1901.04085, 2019.

[6] Microsoft, “Foundry Local Documentation,” Microsoft Learn, 2026. [Online]. Available: https://learn.microsoft.com/en-us/azure/foundry-local/

[7] Microsoft, “Hybrid Search Using Vectors and Full Text in Azure AI Search,” Microsoft Learn, 2026. [Online]. Available: https://learn.microsoft.com/en-us/azure/search/hybrid-search-overview

[8] Microsoft, “Relevance Scoring in Hybrid Search Using Reciprocal Rank Fusion,” Microsoft Learn, 2026. [Online]. Available: https://learn.microsoft.com/en-us/azure/search/hybrid-search-ranking

[9] Microsoft, “Azure OpenAI Service Documentation,” Microsoft Learn, 2026. [Online]. Available: https://learn.microsoft.com/en-us/azure/ai-services/openai/

[10] Microsoft, “Azure Blob Storage Documentation,” Microsoft Learn, 2026. [Online]. Available: https://learn.microsoft.com/en-us/azure/storage/blobs/

[11] Microsoft, “Microsoft OneLake Documentation,” Microsoft Fabric, 2026. [Online]. Available: https://learn.microsoft.com/en-us/fabric/onelake/

[12] S. Ramírez, “FastAPI Documentation,” 2026. [Online]. Available: https://fastapi.tiangolo.com/

[13] SQLite Consortium, “SQLite Documentation,” 2026. [Online]. Available: https://www.sqlite.org/docs.html

[14] UKP Lab and Hugging Face, “Sentence Transformers Documentation,” 2026. [Online]. Available: https://www.sbert.net/docs/

[15] D. Brown, “rank-bm25: A Collection of BM25 Algorithms in Python,” GitHub repository. [Online]. Available: https://github.com/dorianbrown/rank_bm25

[16] J. Singer-Vine, “pdfplumber Documentation and Source Repository,” GitHub repository. [Online]. Available: https://github.com/jsvine/pdfplumber

[17] python-openxml, “python-docx Documentation,” 2026. [Online]. Available: https://python-docx.readthedocs.io/en/latest/

[18] The pandas Development Team, “pandas Documentation,” 2026. [Online]. Available: https://pandas.pydata.org/docs/

[19] Almende B.V. and Contributors, “vis-network Documentation,” 2026. [Online]. Available: https://visjs.github.io/vis-network/docs/network/

[20] M. E. Kolay, “FoundryRAG: Local-Rag-Foundry-assistant,” project repository and evaluation artifacts, 2026. [Online]. Available: https://github.com/MEK-0/Local-Rag-Foundry-assistant