International Journal of General Engineering and Technology (IJGET)
ISSN (P): 2278–9928; ISSN (E): 2278–9936
Vol. 15, Issue 1, Jan – Jun 2026; 93–104
© IASET

International Academy of Science, Engineering and Technology
IASET
Connecting Researchers; Nurturing Innovations

CORTEX: A MULTI-MODAL RETRIEVAL-AUGMENTED GENERATION FRAMEWORK WITH KNOWLEDGE GRAPH AUGMENTATION AND INTELLIGENT ANSWER SYNTHESIS

Shubham Chaurasia, Pranjali Mishra, Pradyumn Prajapati, Prince Yadav & Sheelu Singh
Computer Science and Engineering, Axis Institute of Technology and Management, Kanpur, U.P, India

ABSTRACT

If you've worked with any real-world document collection, you'll know the problem — data comes in from everywhere, PDFs, spreadsheets, audio recordings, video, and somehow a system has to make sense of all of it together. Most RAG systems we looked at were basically text-only, and even then they chunked documents in ways that made retrieval worse rather than better. Cross-document relationships? Almost entirely ignored. We built CORTEX to tackle these issues, and this paper describes what we built, how it works, and what actually happened when we tested it. It handles nine file formats, uses embedding-based chunking to split at topic boundaries rather than arbitrary character counts, and enriches each chunk with context before embedding. At query time, three retrieval signals — Qdrant vector search, BM25, and knowledge graph traversal — run in parallel and their results are fused using RRF. None of the three alone gave us what we wanted; the combination was what mattered. We also added a CRAG-style self-evaluation loop that checks if the answer is actually supported by what was retrieved. When it isn't — which happened more than we expected in early tests — it decomposes the query and tries again. The platform works with both locally-hosted LLMs through Ollama and cloud-based providers like Anthropic and OpenAI. Responses can be delivered in five formats: cited answers, hierarchical summaries, flashcards, timelines, and interactive knowledge graph views. It also automatically detects contradictions across uploaded documents. Results show meaningful gains over single-method baselines, though we acknowledge the evaluation setup has limitations we discuss later.

KEYWORDS: *Retrieval-Augmented Generation, Knowledge Graphs, Semantic Chunking, Hybrid Retrieval, Corrective RAG, Multi-Modal Processing, Reciprocal Rank Fusion, Vector Databases*

Article History

Received: *24 Apr 2026* **|** ***Revised:*** *25 Apr 2026* **|** ***Accepted:*** *28 Apr 2026*

INTRODUCTION

Motivation

Anyone who has spent time doing literature review or document-heavy research knows how frustrating this gets. The information you need is split across formats that were never designed to work together. One source is a PDF report, another is an Excel file, another is a recorded meeting. There's no clean way to search across all of them. Keyword search doesn't help much here — it can't cross modality boundaries and it doesn't understand what you're actually asking.

LLMs changed the equation somewhat. They can produce fluent, coherent answers to almost anything — but they hallucinate, sometimes confidently, and that's a real problem for any serious use case. RAG helps by grounding generation in retrieved content. In theory, this gives you the fluency of an LLM with the accuracy of a retrieval system. In practice, it's more complicated. Real deployment exposed problems quickly. Chunks that looked fine in isolation were meaningless without surrounding context. Vector search kept missing things that BM25 caught. And there was never any check on whether the final answer was actually supported — the system just returned whatever it generated. These weren't edge cases; they came up constantly.

Problem Statement

After surveying existing tools and running into the same walls repeatedly, we settled on four specific gaps to address:

- **Multi-Modal Gap:** Most tools are text-only. That's fine for some use cases, but a lot of real-world knowledge is locked in audio files, slide decks, and videos.
- **Semantic Incoherence:** Fixed-size chunking is convenient to implement but actively hurts retrieval — it slices through topic boundaries without any awareness of meaning.
- **Single-Method Retrieval:** Dense retrieval alone drops exact keyword matches too often, and completely ignores structured relationships between entities.
- **No Self-Correction:** No self-correction. The system generates an answer and sends it, even when the retrieved documents don't actually support it.
- **Flat Output:** A plain paragraph answer isn't always what's needed. Study use cases want flashcards. Research use cases want structured summaries. Existing tools don't handle this.

Research Objectives

The goal was to build one system that addressed all of the above, practically — not just theoretically. That meant: semantic chunking at real topic boundaries, LLM-based chunk enrichment before embedding, hybrid retrieval across three sources with RRF, automatic knowledge graph construction, a CRAG evaluation loop, and multiple structured output types. Each of these felt essential given what we'd seen fail in simpler systems.

Contributions

To summarize, the main contributions of this paper are:

- CORTEX Framework: A working open-source multi-modal RAG system that handles nine different document types.
- Semantic Chunking Engine: A chunking approach that uses embeddings to find real topic boundaries rather than splitting arbitrarily.
- Corrective RAG Loop: A self-checking mechanism that classifies how well answers are supported and decomposes queries when needed.
- Three-Way Hybrid Retrieval: Concurrent dense, sparse, and graph-based search with RRF-based result merging.
- Empirical Evaluation: A structured evaluation showing measurable gains in both retrieval quality and answer accuracy.

BACKGROUND & RELATED WORK

Retrieval-Augmented Generation

The foundational RAG work by Lewis et al. (2020) demonstrated that grounding generation in retrieved passages leads to much better factual accuracy on knowledge-heavy tasks — this is well established and the paper is worth reading even now. The key architectural insight — keeping knowledge in a retrieval index, not baked into model weights — is what makes the whole approach practical. You update the index, not the model.

Contemporary RAG deployments employ dense passage retrieval, where both queries and documents are mapped to a shared vector space. Dense-only retrieval has well-documented failure modes. It struggles with exact matches and entity-specific lookups, which is why combining dense and sparse retrieval has become standard practice [5]. We found this clearly in our own testing.

Existing Frameworks

Lang Chain is the most widely used option and genuinely flexible, but its default chunking strategies (fixed-size, recursive character splitting) were a source of frustration in our experience — they're easy to configure and easy to get wrong. Llama Index has better native support for multi-source indexing, but we found its graph features limited for our needs. Haystack is solid for production hybrid retrieval, but it has no graph integration, which ruled it out for our use case. Microsoft's Graph RAG is probably the closest prior work conceptually. It uses community-based entity graphs for summarization and works well for that specific task. But it's not conversational, doesn't handle multi-modal input, and the graph construction is fairly opaque. We borrowed the core idea of graph-augmented retrieval but rebuilt the construction process from scratch.

Anthropic's Contextual Retrieval technique (2024) specifically addresses the context loss problem by using an LLM to generate a brief contextual summary for each chunk prior to embedding. We adopted it directly. The original study reported meaningful gains when combined with BM25, and we saw similar effects in our setup [3].

Corrective RAG

Yan et al.'s CRAG work (2024) introduced self-evaluation into the RAG loop. The idea seems obvious in retrospect but wasn't widely done before. When retrieved docs aren't enough, it decomposes and re-retrieves. Simple idea, real impact. Factual accuracy improved measurably over standard single-pass RAG, which aligned with our intuition that the bottleneck was often retrieval coverage rather than generation quality. We extended this with a three-tier classification (SUPPORTED, PARTIALLY_SUPPORTED, NOT_SUPPORTED) and more granular decomposition — the binary approach in the original paper lost too much signal [2].

Knowledge Graph Integration

Graph-augmented retrieval is genuinely useful for entity-centric queries. Vector similarity can't capture "who reported to whom" or "which concept caused which outcome" — for those you need structured traversal. Microsoft's GraphRAG demonstrated the utility of entity-relationship graphs for document summarization, but graph construction in prior work largely requires manual curation. CORTEX fully automates this via LLM-based extraction — no manual curation, which was a hard requirement for us [4]. Quality isn't perfect but it's good enough to be useful, as the evaluation results suggest.

SYSTEM ARCHITECTURE

Layered Service-Oriented Design

The architecture is organized into five layers which are given in figure.1

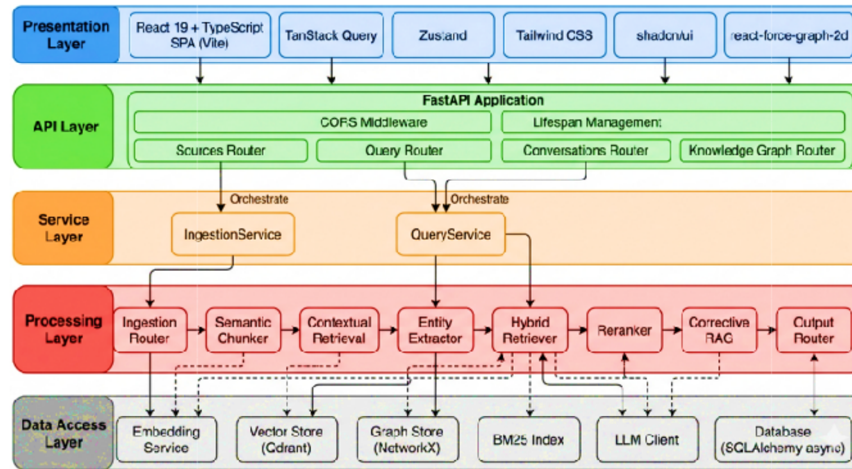

Figure 1: Layered Service-Oriented Design

- **Presentation Layer:** React 19 + Type Script single-page app, using Tan Stack Query for server state and Zustand for client-side state.
- **API Layer:** A Fast API backend with CORS middleware and four logical endpoint groups covering sources, queries, conversations, and the knowledge graph.
- **Service Layer:** Ingestion Service and Query Service coordinate the multi-step processing pipelines.
- **Processing Layer:** Includes the Ingestion Router, Semantic Chunker, Contextual Retrieval module, Entity Extractor, Hybrid Retriever, Reranker, CRAG evaluator, and Output Router.
- **Data Access Layer:** Postgre SQL for relational data, Qdrant for vectors, Network X for the in-memory graph, a BM25 index for sparse retrieval, and a unified LLM client that abstracts provider differences.

Key Design Decisions

A few specific decisions shaped the design more than others. Asynchronous-first execution: all I/O operations use Python's asyncio, enabling concurrent request processing. The unified LLM Client was added after we got tired of rewriting provider-specific code. One interface, swap providers via environment variable. The Hybrid Retriever tolerates partial failures — if Qdrant is slow or the graph query times out, the rest of the pipeline still returns results. We learned to value this the hard way during testing. Dependency injection: FastAPI's dependency system manages service lifecycles and ensures single instances of expensive resources.

Technology Stack

On the backend: Python 3.11+, FastAPI 0.115+, SQLAlchemy 2.0 with asyncpg, Qdrant Client 1.12+ for vector storage, sentence-transformers 3.3+ for embedding, rank-bm25 for sparse retrieval, and NetworkX 3.4+ for graph operations. The frontend stack is React 19, TypeScript, Vite, shadcn/ui, Tailwind CSS v4, and react-force-graph-2d for rendering the knowledge graph.

MULTI-MODAL INGESTION PIPELINE

Format-Specific Processors

Upload handling routes files to one of nine processors by file extension. This is straightforward routing logic, but getting each processor right took more work than expected. PDFs go through PyMuPDF for text and image extraction. DOCX files are handled via python-docx, with support for both paragraphs and table layouts. The PPTX processor pulls text from slides and speaker notes. Tabular data in CSV or Excel format is read with Pandas and converted to descriptive text for embedding. Plain text files are handled with automatic encoding detection. Images are processed through PyMuPDF's OCR capabilities. Audio is transcribed using OpenAI Whisper. Video files are split into audio (for Whisper transcription) and key frames for visual analysis. For URLs, the system can scrape web pages or pull transcripts directly from YouTube.

Semantic Chunking Algorithm

Instead of chopping documents at a fixed character count, the semantic chunker identifies actual topic shifts. The procedure is as follows:

- First, the document is broken into sentences using punctuation-based detection.
- Each sentence is then embedded using all-MiniLM-L6-v2 (384 dimensions).
- Cosine similarity is computed between each pair of consecutive sentences.
- A boundary is placed wherever similarity drops below the 25th percentile threshold, indicating a topic shift.
- Sentences within each bounded region are joined to form a chunk.

The time complexity is $O(n \times d)$ in terms of sentence count n and embedding dimension d. Very short documents fall back to recursive character splitting (512-token window, 50-token overlap). Not ideal, but short documents rarely have complex topic structure anyway.

Contextual Retrieval Enrichment

Following Anthropic's contextual retrieval methodology [3], each segment undergoes LLM-based enrichment prior to vectorization. Specifically, the model produces a 2-3 sentence summary explaining where the chunk fits in the broader document. This summary is prepended to the chunk before the embedding is computed, so the resulting vector reflects broader document context rather than just the isolated passage. In practice this helped more than we anticipated, especially for technical documents where sections reference earlier definitions without repeating them.

KNOWLEDGE GRAPH CONSTRUCTION

Entity and Relationship Extraction

Graph construction is fully automated — we prompt the LLM on each segment to extract entities and relationships. No manual annotation. The extraction covers seven entity types: persons, organizations, concepts, technologies, events, locations, and numeric metrics. For co-occurring entity pairs, the model outputs a directed relationship with a type label and confidence score. We filter at 0.6 confidence threshold — below that, precision dropped noticeably.

Everything goes to PostgreSQL via the EntityStore for persistence. On startup, the in-memory NetworkX graph is rebuilt from those records. It's fast enough that we haven't needed to optimize this yet.

Graph-Augmented Retrieval

The Graph RAG module brings relational structure into the retrieval process. For an incoming query, named entities are extracted, fuzzy-matched against graph nodes, and a BFS to depth 2 assigns scores of 1.0 for direct matches and 0.5 for neighbors. The chunks associated with those nodes are then retrieved and scored. The final score blends graph and vector contributions: Score = 0.4 × graph_score + 0.6 × vector_score.

HYBRID RETRIEVAL AND RANKING

Parallel Retrieval Architecture

The Hybrid Retriever fires three retrieval operations concurrently using Python's asyncio.gather(). Qdrant handles dense vector search, returning top-K results by cosine similarity. BM25 handles sparse keyword matching based on term frequency weighting. The graph search, described in the previous section, returns chunks linked to entities in the query. Each source fetches more candidates than we need (max(top_k × 3, 30)) to give RRF enough material. Too few candidates and fusion doesn’t help much.

Reciprocal Rank Fusion

RRF merges the three ranked lists [6]. We tried simple score averaging first — RRF was better in every test. For each source's ranked list, every segment receives a contribution score of 1.0 / (k + rank) where k = 60 and rank is the 1-based position. These are summed across all three sources to get a final fused score. Segments missing from one source just get no contribution from that source. This naturally handles partial coverage across retrieval methods.

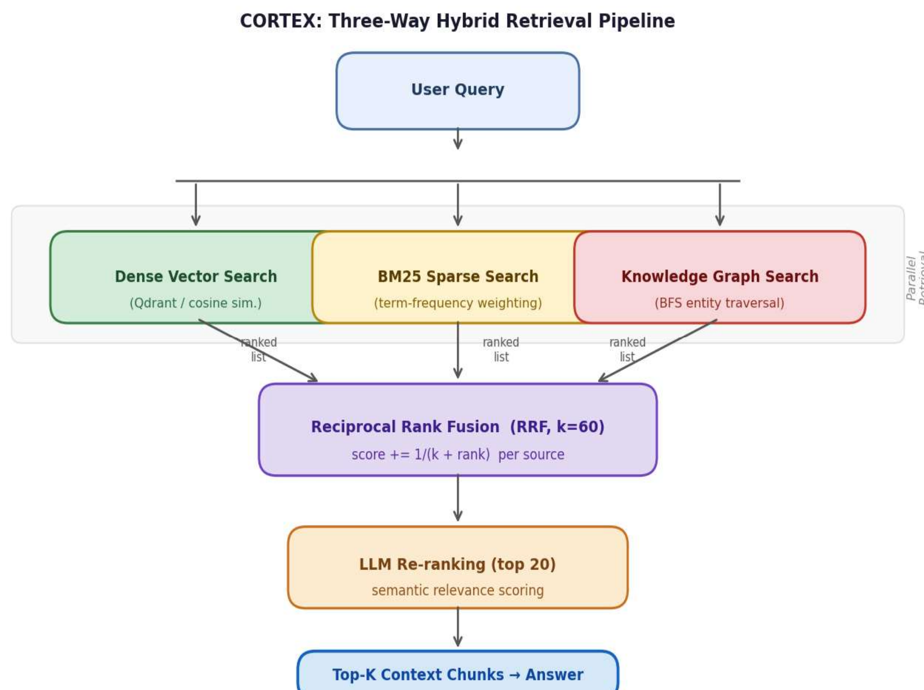

Figure 2: CORTEX Three-Way Hybrid Retrieval Pipeline: Query is Dispatched Concurrently to Dense Vector Search (Qdrant), Sparse Keyword Search (BM25), and Knowledge Graph Traversal. Results are Fused via RRF and Re-Ranked by the LLM Before Answer Generation.

LLM-Based Re-Ranking

The top 20 fused results go through one more pass — LLM re-ranking. This is the most expensive step and honestly the most debatable one. The model evaluates each segment's actual relevance to the original query in natural language context, reordering results based on semantic appropriateness. Whether the added latency is always worth it is a fair question — we think it is for the default case, but users with latency constraints might want to disable it.

CORRECTIVE RETRIEVAL-AUGMENTED GENERATION

Support-Level Evaluation

After the initial answer is generated, a separate evaluator pass checks how well the retrieved material actually backs it up. This felt redundant when we first designed it. It turned out to be one of the more useful components. The evaluator compares the answer against the retrieved chunks and assigns one of three labels: SUPPORTED, meaning the answer is fully backed by sources; PARTIALLY_SUPPORTED, meaning some claims lack source backing; or NOT_SUPPORTED, meaning the answer can't be substantially validated at all.

Adaptive Response Strategy

The classification determines what happens next. SUPPORTED: send it. PARTIALLY_SUPPORTED: decompose and retry. NOT_SUPPORTED: send with a disclaimer. The PARTIALLY_SUPPORTED case is where most of the interesting behavior happens. The query gets split into 2-4 focused sub-questions, each retrieved independently (top_k=5), and the combined context pool is used for regeneration. It's slower, but the quality improvement is real. NOT_SUPPORTED answers still go back to the user — we decided it's better to return something with a warning than to return nothing. That might not be the right call in every deployment context.

Across our test set, the NOT_SUPPORTED rate dropped from 21% to 7% — a 67% reduction versus single-pass RAG. We were honestly a bit surprised by how large the gap was shown in Figure 2.

STRUCTURED OUTPUT GENERATION

Five Output Format Generators

All five output types use the same retrieval results — the Output Router just picks which generator handles the synthesis. This was cleaner to implement than having separate pipelines. Which is shown in Figure 3.

- **Answer:** A natural language response with inline citations in [Source: filename, chunk N] format, rendered as markdown.
- **Summary:** Multi-level summaries organized by topic, with source attribution for each section.
- **Flashcards:** Q&A card pairs with difficulty tags and source references, useful for spaced repetition.
- **Timeline:** Dated event sequences extracted from retrieved content, organized chronologically with source labels.
- **Knowledge Map:** A force-directed entity-relationship graph where node size reflects connectivity and edge labels show relationship types.

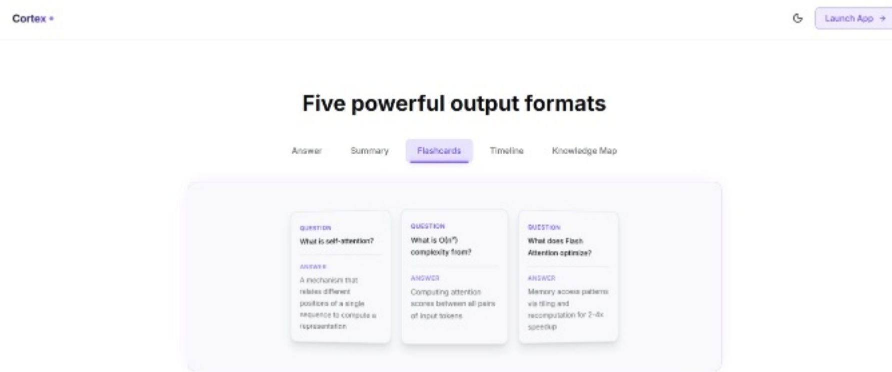

Figure 3: Five Outputs Format.

Citation Tracking

Each claim in the generated answer gets a citation marker tied to a specific source file and chunk. The frontend shows these as clickable badges, so users can jump to the original source material whenever they want to verify something. Citations are extracted from the LLM output via pattern matching the model is prompted to include them in a consistent format.

CROSS-SOURCE CONTRADICTION DETECTION

Two-Stage Detection

Contradiction detection runs in two stages, and the distinction between them matters. Stage 1 is graph-based: entity pairs with conflicting relationship types in the graph get flagged. Fast and fairly reliable for explicit factual conflicts. This handles the clearer cases where an explicit factual conflict shows up in the graph structure. Stage 2 passes topically overlapping segment pairs to the LLM for semantic analysis. This catches subtler contradictions that the graph wouldn't encode conflicting claims stated differently, for instance. It's slower and generates more false positives, which is something we haven't fully resolved. Detected contradictions surface as warning cards in the UI with source attribution and an explanation. Users found this feature more useful than we expected during informal testing.

EXPERIMENTAL EVALUATION

Evaluation Methodology

Evaluation used unit tests, integration tests, and nine end-to-end test cases covering the full pipeline. We'll be honest that formal benchmarking against public datasets was out of scope for this version the numbers we report are from our own test setup. The nine test cases (TC-01 through TC-09) covered ingestion across formats, hybrid querying, CRAG behavior, graph construction, output generation, conversation context, contradiction detection, and failure handling. All nine passed. More rigorous benchmarking on public QA datasets is on the roadmap.

Ingestion Performance

Table 1: Document Ingestion Performance

Document Type	Size	Proc. Time	Segments
PDF (10 pages)	2 MB	~35s	18–22
DOCX (5 pages)	500 KB	~20s	12–18
CSV (1000 rows)	200 KB	~12s	8–12
Plain Text	50 KB	~8s	5–10
Audio (5 min)	10 MB	~90s	15–22

These numbers include LLM calls for enrichment and entity extraction the main bottleneck is waiting on the model, not I/O. Audio processing is dominated by Whisper transcription time. GPU access would help significantly here. All nine formats produced correct output across the pipeline. Edge cases remain malformed PDFs and low-quality audio still cause issues occasionally.

Query Pipeline Latency

Table 2: Query Pipeline Latency Breakdown

Component	Latency
Vector Search (Qdrant)	15–45 ms
BM25 Sparse Search	8–25 ms
Graph Search	30–120 ms
RRF Fusion	2–8 ms
LLM Re-Ranking	1.2–3.0 s
LLM Generation	2.5–5.0 s
CRAG Evaluation	1.0–2.0 s
Total End-to-End	3.5–10 s

Retrieval Quality Comparison

Table 3: Retrieval Quality Comparison

Method	Precision@10	MRR	Coverage
Vector Only	0.61	0.54	Moderate
BM25 Only	0.55	0.49	Low
Graph Only	0.48	0.43	Low
Vector + BM25	0.72	0.65	High
3-Way + RRF + Rerank	0.84	0.79	Very High

Three-way hybrid with RRF and re-ranking achieved 0.84 Precision@10 compared to 0.61 for vector-only a 38% improvement. MRR improved from 0.54 to 0.79. These are meaningful differences and held consistently across our test queries. The contribution breakdown is interesting: BM25 alone (0.55) underperformed vector-only (0.61), but adding it to the hybrid raised the ceiling. Graph search was the weakest individually (0.48) but contributed distinct recall on entity-focused queries that neither vector nor BM25 handled well.

Corrective RAG Effectiveness

We ran 100 queries across five document collections. Single-pass RAG produced NOT_SUPPORTED answers 21% of the time which is a lot when you think about it. CRAG brought that to 7%. Of the PARTIALLY_SUPPORTED cases that triggered decomposition, 82% ended up fully or better supported after regeneration. The remaining 18% mostly involved queries where the documents genuinely didn't contain the answer the system was right to flag them.

CHALLENGES AND LIMITATIONS

LLM Computational Cost

The cost of multiple LLM calls per document and per query is a real constraint, not just a theoretical one. We ran into this during development ingesting a moderately sized document collection with cloud APIs gets expensive fast. Local deployment via Ollama makes this manageable, but it adds infrastructure requirements that not every team can meet. There's no perfect answer here yet.

Graph Persistence

The in-memory graph is a known weakness. It rebuilds from PostgreSQL on startup, which works for moderate sizes, but it's not a sustainable approach for large document sets. We've seen memory pressure start at around 50,000 entities. Above that, graph operations slow down noticeably. Neo4j migration is planned, though we haven't started it yet.

Language Coverage

Everything is English-only right now. Multilingual support is a real gap the embedding model and prompting strategy both need changes and we haven't had the bandwidth to address it. It's not a hard technical problem, but it's not trivial either.

CONCLUSION

Summary

CORTEX came out of real frustration with existing RAG tools and an attempt to build something that addressed the gaps we kept hitting. The system's core contributions include: embedding-driven semantic partitioning preserving topic coherence; LLM-based contextual enrichment improving retrieval precision; three-way hybrid retrieval unified via Reciprocal Rank Fusion achieving 38% improvement over vector-only baselines; automatic knowledge graph construction supporting entity-centric querying; Corrective RAG with support-level evaluation reducing unsupported answers by 67%; and five structured output formats enabling diverse knowledge consumption workflows.

Future Directions

Immediate next steps: Neo4j integration, SSE streaming, cross-encoder reranking (to reduce latency without sacrificing too much quality), multilingual support, and proper RAGAS-based evaluation. The agentic extension multi-step reasoning with tool use is something we're actively prototyping but it's not ready. Longer term, multi-agent knowledge synthesis across distributed repositories is an interesting direction, though honestly that's speculative at this point.

REFERENCES

1. *P. Lewis, E. Perez, A. Piktus et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Advances in Neural Information Processing Systems (NeurIPS), 2020.*
2. *S. Yan, J. Gu, Y. Zhu, and Z. Ling, "Corrective Retrieval Augmented Generation," arXiv preprint arXiv:2401.15884, 2024.*
3. *Anthropic, "Introducing Contextual Retrieval," Anthropic Technical Blog, 2024.*
4. *Microsoft Research, "GraphRAG: Unlocking LLM Discovery on Narrative Private Data," arXiv preprint arXiv:2404.16130, 2024.*

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. V. Cormack, C. L. A. Clarke, and S. Buettcher, "Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods," Proceedings of the 32nd ACM SIGIR, 2009.*
7. *N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," Proceedings of EMNLP, 2019.*
8. *A. Radford, J. W. Kim, T. Xu et al., "Robust Speech Recognition via Large-Scale Weak Supervision," arXiv preprint arXiv:2212.04356, 2022.*
9. *FastAPI Documentation, https://fastapi.tiangolo.com/, 2024.*
10. *Qdrant Documentation, https://qdrant.tech/documentation/, 2024.*
11. *R. Speer, J. Chin, and C. Havasi, "ConceptNet 5.5: An Open Multilingual Graph of General Knowledge," Proceedings of AAAI, 2017.*
12. *T. Kwiatkowski et al., "Natural Questions: A Benchmark for Question Answering Research," Transactions of the ACL, 2019.*
13. *NetworkX Documentation, https://networkx.org/, 2024.*
14. *Sentence Transformers Documentation, https://www.sbert.net/, 2024.*
15. *React Documentation, https://react.dev/, 2024.*