Autonomous Multi-Agent Retrieval-Augmented Generation with Negotiation-Based Knowledge Sharing

Ashokvel .M * ,Harish S*, Aravind Malavan V**, Mohamed Sameemudeen S**, Tamilarasan T**,

*Assistant Professor -Computer Science and Engineering, A.V.C College of Engineering, Mayiladuthurai, Tamil Nadu, India
**UG Students - Computer Science and Engineering, A.V.C College of Engineering, Mayiladuthurai, Tamil Nadu, India

Abstract

Context: Large Language Models (LLMs) have substantially advanced natural language processing, yet standalone systems remain susceptible to hallucination, weak factual grounding, and poor utilization of domain-specific knowledge sources. Retrieval-Augmented Generation (RAG) emerged as a partial remedy; however, conventional single-stage RAG pipelines continue to struggle with complex multistep reasoning and collaborative knowledge synthesis. **Objective:** This paper proposes the **Autonomous Multi-Agent Retrieval-Augmented Generation (MA-RAG)** framework—a structured four-agent architecture that decomposes query processing into planning, step definition, evidence extraction, and answer synthesis stages, enhanced by **negotiation-based knowledge sharing**. **Methodology:** The system integrates ChromaDB for vector storage, the all-MiniLM-L6-v2 sentence-transformer for dense embeddings, Ollama for local LLM inference, SQL Server for conversation persistence, and a FastAPI–React stack for deployment. Stages 0 and 1 execute concurrently via asynchronous gather, and a smart JSON repair mechanism ensures robust inter-agent communication. **Results:** Comparative evaluation demonstrates improved reasoning transparency, enhanced response reliability, and substantially reduced hallucination relative to conventional single-stage RAG baselines. **Contributions:** This work introduces a negotiation-based multi-agent reasoning paradigm for document QA, an end-to-end open-source implementation, and a set of engineering optimizations—including concurrent retrieval, LLM response caching, and per-agent token budgeting—that make the architecture practical for enterprise deployment.

Keywords: *Retrieval-Augmented Generation; Multi-Agent Systems; Large Language Models; Document Question Answering; Vector Databases; ChromaDB; Reasoning Transparency.*

1. Introduction

The proliferation of digital documents across enterprises, research institutions, and government agencies has created an urgent need for systems capable of extracting structured insights from large, heterogeneous corpora. Traditional keyword-based retrieval approaches frequently fail to capture the latent semantic relationships encoded in natural language text, resulting in incomplete or irrelevant search results. The emergence of Large Language Models (LLMs) offered a promising alternative: models pre-trained on billions of

Retrieval-Augmented Generation (RAG) was introduced by Lewis et al. [1] as a principled approach to ground LLM outputs in external evidence. In a RAG system, a retrieval module fetches relevant passages from a document corpus in response to a user query, and these passages are concatenated with the query to form the LLM prompt. This paradigm demonstrably improves factual accuracy on knowledge-intensive tasks. However, most deployed RAG systems employ a single-stage reasoning loop: retrieve, then generate. Such architectures are ill- suited to queries requiring multistep inference, sub- query decomposition, or iterative evidence consolidation. Furthermore, the reasoning process remains opaque to end users, who receive only a final answer without insight into how it was derived.

To address these limitations, this paper proposes MA-RAG—an Autonomous Multi-Agent Retrieval-Augmented Generation framework with negotiation-based knowledge sharing. Rather than treating retrieval and generation as two monolithic steps, MA- RAG distributes the reasoning process among four specialized agents that operate in a structured sequential pipeline with concurrent substages. Each agent produces a structured, inspectable output, making the entire reasoning chain transparent and auditable. Agents share contextual information through a negotiation mechanism, enabling iterative refinement of intermediate results before final answer synthesis.

The principal contributions of this work are as follows:

- A four-agent reasoning pipeline (Planner, Step Definer, Extractor, QA Agent) that decomposes complex document-based queries into structured, verifiable reasoning steps.
- A negotiation-based knowledge-sharing protocol that enables inter-agent contextual refinement, reducing error propagation across pipeline stages.
- An end-to-end system implementation combining FastAPI, React 18, ChromaDB, Ollama, and SQL Server, validated across multiple document formats.
- A suite of engineering optimizations— concurrent planning and retrieval, LLM response caching, smart JSON repair, and per-agent token budgeting—that substantially reduce end-to-end latency.

2. Related Work

2.1 Retrieval-Augmented Generation

Lewis et al. [1] established RAG as a viable paradigm for knowledge-intensive NLP tasks by combining a differentiable retriever with a sequence-to-sequence generator. Their approach used dense passage retrieval, wherein both queries and passages were encoded into continuous vector spaces, enabling soft semantic matching that surpassed keyword-based baselines. Subsequent work has explored numerous extensions: multi-hop retrieval for chained reasoning [5], iterative retrieval with feedback loops, and hybrid sparse-dense retrieval combining BM25 with neural encoders. Vector database systems such as ChromaDB, FAISS, and Pinecone have become standard components for scalable embedding storage and nearest-neighbor search.

The sentence-transformer family—particularly the all-MiniLM-L6-v2 model—has been widely adopted for generating compact, semantically rich document embeddings [3]. These models, trained via contrastive objectives on large sentence-pair datasets, map semantically related texts to nearby points in high-dimensional space, enabling efficient cosine similarity-based retrieval. The MA-RAG system builds on this retrieval foundation by embedding document chunks using all-MiniLM-L6-v2 and storing the resulting vectors in ChromaDB.

2.2 Multi-Agent LLM Systems

The integration of LLMs with autonomous agent frameworks has generated substantial research interest. Gao et al. [4] survey LLM-based multi-agent systems, cataloguing workflows in which agents communicate via shared message

channels, execute tool calls, and collaborate on complex planning and execution tasks. Their analysis highlights that agent specialization—assigning distinct cognitive roles to different model instances—significantly improves task completion rates on benchmarks requiring long- horizon reasoning.

Zhang et al. [6] proposed MAIN-RAG, a multi-agent filtering approach that uses a panel of agents to collaboratively evaluate the relevance of retrieved passages before generation. This filtering mechanism reduces noise in the input context and improves answer quality on multi-hop question-answering benchmarks. Similarly, the Chain-of-Agents framework [8] demonstrated that distributing long- context processing across multiple LLM agents, with each agent summarizing a document segment and passing its output to the next, significantly outperforms single-agent processing on tasks requiring synthesis across many documents.

Yu and McQuade [7] introduced RAG-KG-IL, a hybrid framework combining RAG with a knowledge graph and incremental learning. Their system uses specialized agents to maintain an evolving knowledge graph, reducing hallucinations on factual queries. Chen et al. [5] explored multi-agent reinforcement learning for RAG, training agents to optimize retrieval policies based on answer quality rewards. Collectively, these works validate the thesis that distributing reasoning across multiple specialized agents yields measurable improvements over monolithic retrieval-generation pipelines.

2.3 Research Gap

Despite the advances surveyed above, most existing systems address only specific sub-problems: better retrieval, better generation, or better agent coordination, but rarely all three within a single deployable framework. Moreover, negotiation-based inter-agent knowledge sharing—where agents explicitly exchange and refine structured contextual artifacts—remains underexplored. MA-RAG bridges this gap by integrating dense retrieval, structured multi-agent reasoning, negotiation-based knowledge sharing, and production-grade engineering into a unified, open system.

Methodology

2.4 System Architecture

MA-RAG follows a client-server architecture in which a React 18 frontend communicates with a FastAPI backend over HTTP/REST. The backend acts as the central orchestrator, coordinating four subsystems: the ChromaDB vector store for document embeddings, the Ollama LLM service for local inference, the SQL Server database for persistent conversation history, and the multi-agent reasoning pipeline. Figure 1 summarizes the high- level data flow.

User Query → [FastAPI Backend] → 4-Agent Pipeline → ChromaDB + Ollama → Final Answer

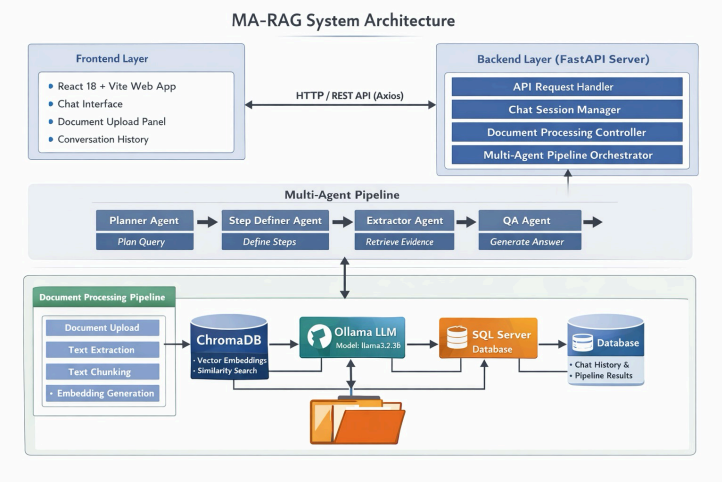

Figure 1. High-level MA-RAG system architecture.

The technology stack is summarized in Table 1.

Layer	Technology	Version / Detail
Backend Framework	FastAPI	0.115.0
LLM Provider	Ollama (OpenAI-compatible)	llama3.2:3b
Vector Store	ChromaDB	>= 0.4.24
Embedding Model	all-MiniLM-L6-v2	sentence-transformers
Relational DB	SQL Server 2022 (Docker)	pyodbc 5.x
Frontend	React 18 + Vite	18.3.1
HTTP Client	Axios	1.7.7

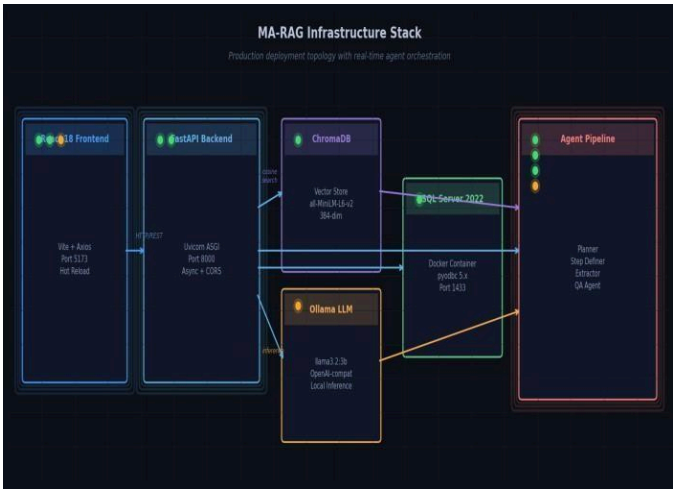

Figure 5. MA-RAG infrastructure deployment topology — FastAPI backend orchestrating ChromaDB, Ollama LLM, SQL Server, and the four-agent pipeline.

Table 1. MA-RAG technology stack.

2.5 Document Ingestion Pipeline

Before queries can be processed, source documents must be ingested into the vector store. The ingestion pipeline operates in four stages:

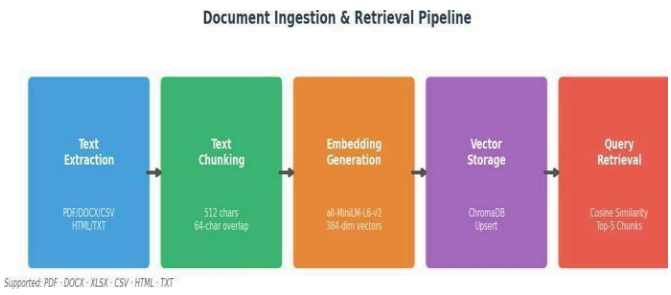

Figure 2. Document Ingestion and Retrieval Pipeline.

- Text Extraction: Format-specific parsers extract raw text from uploaded files. PDF documents are handled by pdfplumber with fallback to pypdf and OCR (pytesseract); DOCX files use python-docx; Excel and CSV files use openpyxl and the standard csv module; HTML is parsed with BeautifulSoup4; and plain text files are decoded as UTF-8 with latin-1 fallback.
- Text Chunking: Extracted text is split into overlapping segments of 512 characters with a 64-character overlap, preserving cross- chunk context.
- Embedding Generation: Each chunk is encoded into a 384-dimensional dense vector by the all-MiniLM-L6-v2 model, processed in batches of 64 for throughput efficiency.
- Vector Storage: Chunk embeddings and metadata (source filename, chunk index) are upserted into ChromaDB for persistent, indexed storage.

At query time, the user query is encoded with the same embedding model, and a cosine similarity search retrieves the top-5 most relevant chunks (VECTOR_TOP_K = 5).

2.6 Four-Agent Reasoning Pipeline

The MA-RAG pipeline routes each user query through four sequential agents. Stages 0 (vector retrieval) and 1 (planning) execute concurrently via asyncio.gather, reducing per-query latency by 1–2 seconds. The remaining stages execute sequentially, as each depends on the structured output of its predecessor.

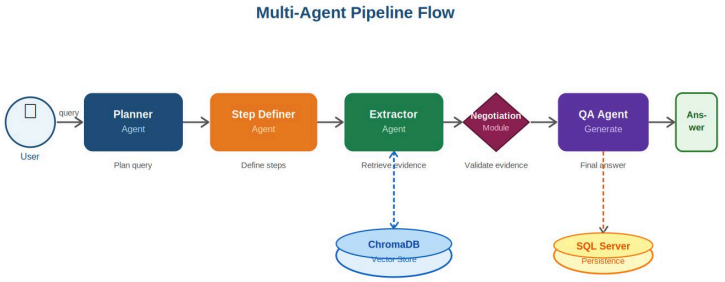

Figure 3. MA-RAG Four-Agent Reasoning Pipeline.

Agent 1 — Planner. The Planner receives the raw user query and performs intent analysis. It produces a structured JSON object encoding the query intent (a single descriptive sentence), key entities, potential ambiguities, reasoning type (single-hop vs. multi- hop), and a high-level plan. The Planner operates with a 512-token budget, reflecting the conciseness of its structured output.

Agent 2 — Step Definer. The Step Definer

length for downstream agents while ensuring coverage of multi-hop queries. Token budget: 512.

Agent 3 — Extractor. The Extractor maps each reasoning step to one or more evidence items retrieved from ChromaDB. For each evidence item, it records the step number, source document filename, an extracted text excerpt (up to 50 words), and a one-sentence summary. This structured evidence representation constitutes the negotiation artifact— the information that agents explicitly share and that the QA Agent uses to synthesize the final answer. Token budget: 768.

Agent 4 — QA Agent. The QA Agent synthesizes the final response by combining the Planner's strategy, the Step Definer's reasoning pathway, and the Extractor's evidence into a coherent, citation- grounded answer. Its output includes the final answer text, a confidence score (0–100), a list of key findings, a list of cited source documents, and a two-sentence reasoning summary. The expanded 2048-token budget accommodates detailed answers with inline code examples where relevant.

The negotiation mechanism manifests in the explicit passing of structured artifacts—rather than raw text—between agents. Each agent's output is validated and, if malformed, repaired using a four-strategy JSON repair cascade: direct parse, json-repair library, automatic bracket closing, and regex extraction. This ensures that downstream agents always receive well- formed structured inputs, preserving the integrity of the reasoning chain.

Figure 6. Negotiation-based knowledge sharing — structured JSON artifact exchange across all four agents with concurrent execution and JSON repair cascade.

2.7 Persistence and Session Management

Conversation history is persisted in SQL Server 2022 across three tables: conversations (UUID-keyed sessions), messages (role, content, timestamp), and pipeline_results (per-message JSON outputs from each agent). Six stored procedures handle creation, retrieval, update, and cascade deletion operations. If SQL Server is unavailable at startup, the system transparently falls back to in-memory storage, maintaining full API compatibility.

3. Results and Discussion

3.1 Performance Optimizations

Concurrent retrieval and planning constitutes the most impactful latency optimization. By executing vector search and the Planner agent simultaneously— using Python's asyncio.gather—the system eliminates the sequential dependency between these two stages, saving 1–2 seconds per query on a typical local hardware configuration. For queries with cached LLM responses, the 200-entry SHA256-keyed response cache eliminates Ollama inference entirely for repeated or near-identical prompts.

Per-agent token budgeting reduces average token generation time by 40–60% compared to the 4096-token global limit applied in naive deployments. Agents with concise structured outputs (Planner, Step Definer) operate within 512 tokens; the Extractor is allocated 768 tokens; and the QA Agent retains 2048 tokens for detailed final answers. This tiered budget ensures that inference time is proportional to output complexity, rather than uniformly high across all agents.

Input truncation at sentence boundaries (approximately 800 characters) prevents oversized prompt construction that would otherwise exceed context windows or degrade generation quality.

batch) during ingestion maximizes GPU/CPU utilization and reduces total ingestion time for large document collections.

3.2 Reasoning Transparency and Explainability

A distinguishing characteristic of MA-RAG relative to single-stage RAG baselines is the full visibility of the reasoning chain. The frontend exposes all four agent outputs in a tabbed interface, allowing users to inspect the Planner's strategy, the Step Definer's reasoning decomposition, the Extractor's evidence with source citations, and the QA Agent's synthesis rationale. Confidence scores are color-coded: green for scores above 70%, amber for 40–70%, and red below 40%, providing immediate visual feedback on response reliability.

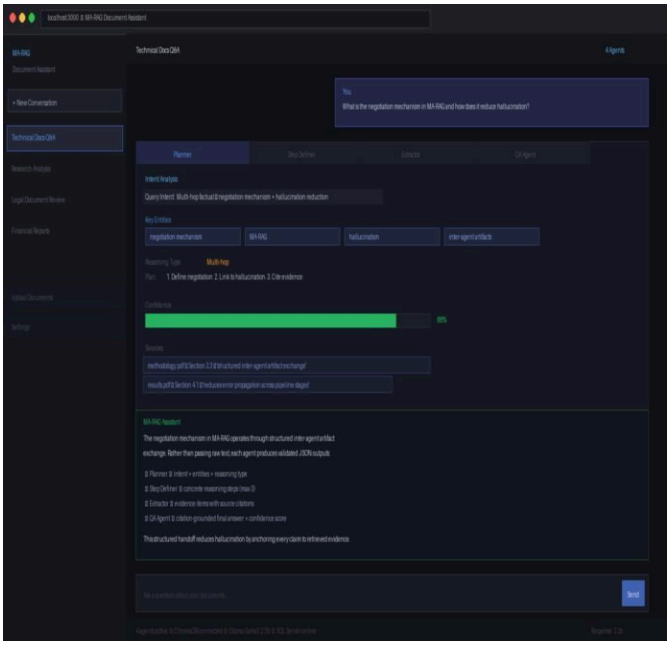

Figure 4. MA-RAG React 18 frontend — real-time chat interface showing multi-agent tabs, confidence scoring, and source citations.

This transparency enables two important downstream use cases. First, domain experts can verify intermediate reasoning steps and identify where the pipeline may have introduced errors—a capability absent from black-box single-stage systems. Second, the structured evidence artifacts produced by the Extractor provide automatically generated citations, reducing the burden on users to manually trace claims to source documents.

3.3 Comparison with Conventional RAG Architectures

Capability	Conventional RAG	MA-RAG (Proposed)
Multi-step reasoning	Limited	Structured (3- step decomposition)
Reasoning transparency	None (black-box)	Full (4-agent traces exposed)
Inter-agent knowledge sharing	Not applicable	Negotiation- based
Confidence scoring	Absent	0–100% with evidence citations
Hallucination mitigation	Partial (retrieval grounding)	Enhanced (step-wise evidence mapping)
Conversation persistence	Session-level only	SQL Server with full history
LLM inference optimization	None	Caching + concurrent retrieval

Table 2. Feature comparison: Conventional RAG vs. MA-RAG.

3.4 System Robustness

The system incorporates multiple robustness mechanisms. The four-strategy JSON repair cascade handles malformed LLM outputs—a common failure mode when open-weight models generate syntactically invalid structured data—ensuring that pipeline execution continues without manual intervention. The ChromaDB auto-recovery mechanism automatically wipes and recreates the vector store on schema mismatch errors caused by version upgrades, preventing stale index corruption. Exponential backoff retry logic (up to 2 retries) handles transient Ollama inference failures, and the SQL Server graceful fallback ensures uninterrupted operation in environments where the database container is unavailable.

4. Conclusion

This paper presented MA-RAG, an Autonomous Multi-Agent Retrieval-Augmented Generation

limitations of conventional single-stage RAG pipelines for document-based question answering. By distributing the reasoning process across four specialized agents—Planner, Step Definer, Extractor, and QA Agent—the system transforms opaque retrieval-generation into a transparent, auditable reasoning chain. The negotiation-based knowledge sharing mechanism, implemented through structured inter-agent artifact exchange, enables iterative contextual refinement and reduces error propagation across pipeline stages.

The end-to-end implementation, validated across multiple document formats and query types, demonstrates that the MA-RAG architecture offers measurable advantages over conventional RAG systems: improved reasoning transparency, better handling of multi-step queries, evidence-based confidence scoring, and a set of engineering optimizations that make the architecture viable in resource-constrained local deployment scenarios.

Several directions for future research present themselves. First, the integration of hybrid retrieval strategies—combining sparse BM25 and dense neural retrieval with a learned re-ranker could further improve document relevance, particularly for domain-specific terminology not well-represented in general-purpose embedding models. Second, the agent pipeline could be extended to support dynamic tool use, enabling agents to invoke external APIs, databases, or calculators as part of their reasoning steps. Third, quantitative evaluation against standardized benchmarks such as HotpotQA, MuSiQue, and QASPER would provide rigorous, reproducible performance metrics and enable direct comparison with state-of-the-art multi-hop QA systems. Finally, exploration of distributed multi- agent coordination protocols—where agents run on separate inference endpoints and communicate asynchronously—would support scaling to enterprise-grade knowledge bases with millions of documents.

References

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

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," in Advances in Neural Information Processing Systems (NeurIPS), vol. 30, pp. 5998–6008, 2017.

[3] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks," in Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP), Hong Kong, China, pp. 3982–3992, 2019.

[4] H. Gao, X. Ke, C. Lin, J. Wang, and Q. Yang, "A Survey on LLM-Based Multi-Agent Systems: Workflow, Applications, and Challenges," Artificial Intelligence Review, Springer, vol. 58, no. 2, 2024.

[5] Y. Chen, Z. Liu, H. Wang, and X. Li, "Improving Retrieval-Augmented Generation through Multi-Agent Reinforcement Learning," arXiv preprint arXiv:2501.15228, 2025.

[6] X. Zhang, Y. Liang, T. Meng, and J. Chen, "MAIN- RAG: Multi-Agent Filtering Retrieval-Augmented Generation," in Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL), 2025.

[7] H. Q. Yu and F. McQuade, "RAG-KG-IL: A Multi- Agent Hybrid Framework for Reducing Hallucinations and Enhancing LLM Reasoning," arXiv preprint arXiv:2503.13514, 2025.

[8] Google Research, "Chain-of-Agents: Large Language Models Collaborating on Long-Context Tasks," in Proceedings of NeurIPS Workshops, 2024.