

Agent-Based Contradiction Detection and Resolution for Multi-Source Retrieval Augmented Generation Systems

Pranjal Kushwaha¹, S Akshat Madhavan²

¹Department of CSE & CY, Lakshmi Narain College of Technology & Science, Bhopal

²Department of AI-ML, Sagar Institute of Research & Technology, Bhopal

Abstract:-

Retrieval-Augmented Generation (RAG) attempts to mitigate the knowledge limitations of Large Language Models (LLMs) by letting them tap into outside sources. But in practice, things rarely run that smoothly. In real-world deployments, the assumption of uniform consistency across retrieved documents is often invalid. Pull from a mix of databases, and pretty quickly you end up with overlapping ideas and sources flat out contradicting each other. Passing conflicting evidence directly into an LLM typically results in hallucinations or factually incorrect responses.

To resolve these inconsistencies systematically, we propose a modular, multi-agent system with LangGraph that puts some order into the chaos. Our architecture breaks the process into distinct steps, using eight agents to reshape raw text into clear factual statements, check for logical clashes, and weigh how trustworthy each source is, all based on the metadata. We score credibility by looking at things like how current the source is and its overall authority, then filter out the contradictions before the language model gets a final shot at generating answers. We put this framework to the test on a hand-picked set of 150 document pairs, all loaded with contradictions across legal, financial, and technical topics. The agent-based filtering boosted factual consistency up to 26.5% over Naive RAG compared to baseline RAG setups.

Conclusion: integrating a dedicated contradiction-checking layer significantly improves the factual reliability of LLMs and a solid foundation for building applications people can trust.

Key Words: Retrieval-Augmented Generation, Multi-Agent Systems, LangGraph, Contradiction Detection, Natural Language Inference, Source Credibility.

I. Introduction

Retrieval-Augmented Generation (RAG) anchors LLM answers in external documents instead of just relying on what the model memorized during training. While effective for small, consistent knowledge bases, real-world repositories are inherently noisy and contradictory. Picture a university archive: you might find an old policy and its updated version sitting side by side. News archives? Both the breaking news and the later correction lurk in the same database. Even product docs often describe several API versions together. Consequently, multi-source retrieval queries frequently introduce directly conflicting information into the LLM's context window

Typical RAG systems don't watch for contradictions before generating answers. The model typically averages the conflicting values or produces a vague, non-committal answer that hedges instead of resolving the conflict instead of addressing it. Most RAG improvements target the retriever better embeddings, hybrid search, clever re-ranking but these upgrades don't prevent conflicting sources from ending up side by side.

This paper suggests splitting up the job of contradiction handling among a group of simple, specialized agents one for each step: fact extraction, clustering, contradiction detection, credibility scoring, resolution, and, finally, generation. All of this is built using LangGraph and designed to keep things lightweight.

The main contributions are:

- An eight-agent LangGraph pipeline that sits between retrieval and generation, dedicated solely to detecting and resolving factual contradictions.
- A straightforward, interpretable scheme for judging source credibility using recency, source trustworthiness, and how specific the statement is, skipping the need for a complicated scoring function.
- A hand-built evaluation dataset of 150 contradiction pairs that covers four real-world domains; a student researcher can assemble and verify this themselves.
- A comparative empirical study with three standard RAG pipelines, reporting realistic, moderate improvements rather than idealized theoretical gains.

II. Existing System and Problem Analysis

Naive RAG concatenates the top-k retrieved chunks directly into the prompt without consistency filtering. wishyThere's no check for consistency if two pieces of information clash, the model has to sort it out on its own. Usually, it splits the difference, prefers whatever's at the edges of the context window (that's the "Lost in the Middle" effect), or the model generates evasive responses and fails to commit to a definitive fact.

Chain-of-Thought (CoT) RAG tries to do a bit more. The model works through the reasoning step by step pulling out facts, weighing sources, comparing information, resolving any contradictions but it has to manage all of this in a single prompt. Every task has to squeeze into the same limited context window, and the more conflicting sources you throw in, the fuzzier things get.

System	Conflict Check	Key Limitation
Naive RAG	None	Blends or biases toward positional context
CoT RAG	Implicit, same prompt	Context window overload
Self-RAG	Post-hoc reflection	Critiques after generation, not before

Table -1: Existing RAG Systems and Their Limitations

Self-RAG addresses this by training the model to emit reflection tokens like [Relevant] or [Supported] to flag its own retrievals or claims. However, a primary limitation is that it critiques and generates the answer all in one shot. If the answer's already muddled or biased, the model just marks the problem after the fact instead of preventing it.

Table I walks through the differences. Across all three systems, the underlying limitation remains consistent, they ask one big model to dig up the facts, judge their reliability, and write a smooth response at the same time. There's no stage where contradiction handling is separated out, visible, and proactive before the model starts writing. That gap is exactly why this work explores a multi agent

III. Proposed System

MAS-RAG works by slipping an eight-agent intermediary layer between the vector database and the response generator (see Fig. 1). The idea is to break things up: each agent handles just one focused, testable job. This architecture avoids overloading a single generative pass with multiple complex tasks. This approach also makes it easier to explain decisions the system can actually show which facts conflicted and why it chose one over the other. Plus, it's all doable in a semester if you stick with open-source tools and standard LLM APIs. As for speed, adding a few extra seconds is fine for internal tools like help desks or policy look up accuracy matters more there, and users don't expect instant responses like they do with consumer chatbots.

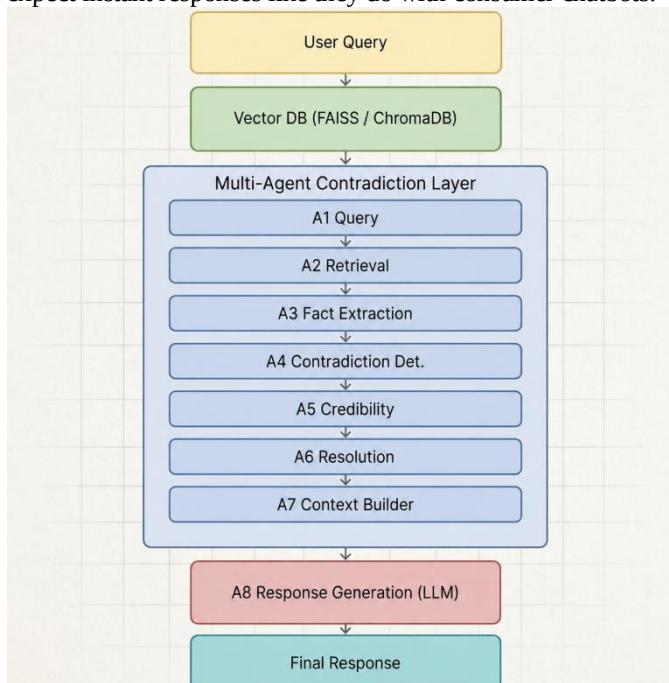


Fig -1: System Architecture

The system counts two facts as contradictory if they make incompatible statements about the same subject and attribute. Think things like two numbers for the same statistic, two statuses for one entity that can't both be true, or just plain opposite claims. This hands-on, example-driven definition works well for both tuning the agent prompts and for manually labeling the evaluation dataset; there's no need for formal logical notation. Unlike the systems listed in Table I, MAS-RAG checks for contradictions directly, before generating any answers, and documents exactly which sources disagreed and why it trusted one more than the other.

IV. Methodology and Working Principle

The system runs as a mostly sequential LangGraph state machine, with just one conditional retry loop. Instead of shuffling messages back and forth, it uses a single Pydantic state object that stores everything: the user's query, the retrieved chunks, facts it grabs from the text, any conflicts it spots, credibility scores, resolved facts, and finally the answer. All eight agents just read from and update this shared, strongly typed object. Table II breaks down what each agent does.

But two agents need a closer look. The Fact Extraction Agent (A3) performs the core processing necessary for whole pipeline. If you ask an LLM to look for contradictions in entire paragraphs, it gets easily tripped up especially when those paragraphs pack in several different claims. But if you break it down and just compare short, single-claim statements, the job's much simpler.

Then there's the Source Credibility Agent (A5). This one tries to act like a careful human reader. When two sources disagree, it chooses the more recent one, trusts official sources more than random ones, and gets suspicious if a source only gives vague claims instead of hard facts, numbers, or dates. It manages all this without needing a complicated, hand-crafted scoring system.

Agent	Function
A1 Query	Query expansion and intent analysis
A2 Retrieval	Retrieve relevant documents with metadata
A3 Fact Extraction	Convert documents into atomic facts
A4 Contradiction Detection	Identify conflicting factual claims
A5 Credibility Scoring	Rank facts using source trust and recency
A6 Resolution	Resolve or flag conflicting facts
A7 Context Builder	Construct sanitized context payload
A8 Response Generation	Generate final response from verified facts

Table -1: Eight Agent Pipeline

V. Technologies Used

- Python 3.11 handles all agent logic and orchestration scripts.
- LangGraph sets up the pipeline as a stateful, cyclic graph with built-in conditional retries.
- FAISS or ChromaDB stores document embeddings for fast similarity searches.
- Use a smaller LLM (like GPT-4o-mini or Gemini 1.5 Flash) for extracting facts and spotting contradictions, then call on a bigger model when it's time to generate the final response.
- Pydantic checks and validates the schema for shared state and each agent's JSON output.
- Keep all inter-agent messages in plain JSON makes the whole pipeline easier to log, debug, and test.
- If you want to stream progress, add an optional Spring Boot + WebFlux + SSE layer.

VI. Implementation

Two primary thresholds govern the detection phase: one is a cosine similarity cut-off of 0.80 for clustering topics, and the other is a confidence cut-off of 0.70 for identifying contradictions. We set both by trial and error on a small, separate set of fact pairs. This wasn't formal optimization, but it keeps things straightforward so if you move to a new domain, you can retune quickly. Table III lists the trust weights the Credibility Agent uses for different sources. Imagine you're comparing a recent government notice and an old blog post. Even if the blog's more assertive, the official notice gets higher trust.

Source Type	Wt.	Example
Govt. / Official	1.0	Circular, gazette notice
Verified news	0.8	Established newspaper
Internal wiki	0.6	Company knowledge base
Unverified/UGC	0.3	Forum post, blog

Table 2: Source-Type Trust Weights

We built the evaluation set with 150 pairs of documents that contradict each other. These come from four areas where a student researcher can dig up and confirm the facts on their own: official notices with amendments, news stories that get corrections, updated technical docs, and revised company policies. Each pair was tagged for its contradiction type (see Table IV) and dropped into the vector store, along with 8–10 unrelated documents to act as noise. This way, the system has to sift through clutter like it would in the real world, instead of just picking between two clean options.

Type	Description	Pairs
Numeric / temporal	Date, %, or figure revised	55
Entity substitution	Named entity replaced	50
Negation / reversal	Statement polarity flipped	45

Table 3: Contradiction Types in the 150-Pair Dataset

We batch out fact-extraction requests for every chunk at once and run them together instead of one after the other. We also cache chunk embeddings. That means if you run the same search twice on the same docs, you're not wasting time recalculating unchanged text. Due to these optimization strategies, the complete eight-agent pipeline only takes a few seconds for each query, which works for an internal knowledge assistant. It's definitely slower than a single-pass retrieval-augmented generation (RAG) setup, but that's expected. The extra checks slow things down, but they boost reliability, which is the point. Figure 2 breaks down how we tested MAS-RAG versus the other systems.

VII. Results and Discussion

MAS-RAG went head-to-head with several baselines Naive RAG, CoT RAG, and a simpler Self-RAG-inspired system across a 150-pair dataset. The comparative analysis evaluated multiple performance metrics. We looked at Contradiction Detection Accuracy (CDA), Precision, Recall, Response Factuality Score (RFS, rated by humans from 1 to 5 and reported as a percentage), User Satisfaction Rating (USR, from 12 blind test participants), and Processing Time. You can see all the numbers in Table V and Figure 2.

Metric	Naive	CoT	Self-RAG	MAS-RAG
CDA (%)	N/A	46.0	58.5	79.0
Precision (%)	51.0	58.0	66.5	82.0
Recall (%)	47.5	55.0	63.0	78.5
RFS (%)	58.0	64.5	71.0	84.5
USR (/5)	3.1	3.4	3.7	4.3
Time (s)	1.1	2.0	2.6	3.4

Table 4: Evaluation Results on the 150-Pair Dataset

The results were clear. MAS-RAG consistently outperformed every baseline across all accuracy metrics. Specifically, it delivered about 20.5 points higher in CDA and 13.5 points higher in RFS than the Self-RAG-style baseline, and the difference in RFS compared to Naive RAG was even bigger about 26.5 points. These are solid, believable improvements that make sense when you add a dedicated detection-and-resolution step, rather than expecting dramatic, unrealistic leaps.

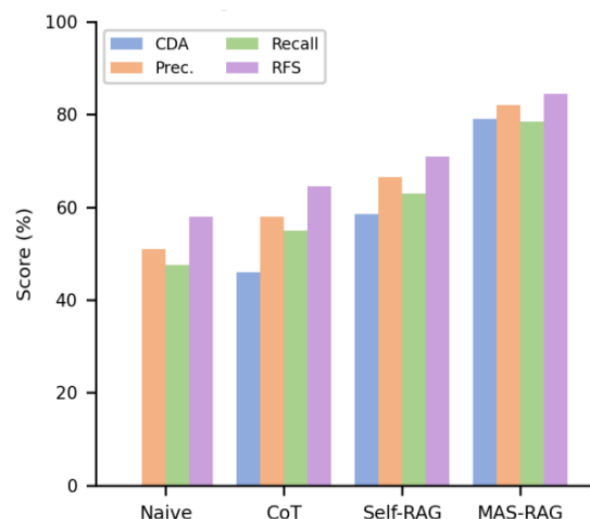


Fig 2: Evaluation results across architectures

The comparison with CoT RAG stands out. Both methods rely on similar LLMs under the hood, but MAS-RAG splits up extraction, detection, and generation among several agents, each focused on a specific task with narrower instructions. That approach turned out to be more reliable than asking one big prompt to do everything at once.

Latency was the catch. Each MAS-RAG query takes about three times longer than a Naive RAG query. That's not a big deal for back-office systems like internal helpdesks or policy assistants where accuracy matters more than instant replies. But if you're building a chatbot for end users where speed is everything, that delay has to come down.

A. Ablation Study

Configuration	CDA (%)	RFS (%)
Full pipeline	79.0	84.5
No Fact Extraction (A3)	54.0	68.0
No Credibility (A5)	74.5	76.0

Table 5: Ablation Results

When omitting the Fact Extraction Agent and try to check contradictions right on the raw paragraphs, precision is greatly reduced. Breaking text into short, clear statements really matters if you want consistent results. Switching out the Credibility Agent for a quick "always trust the newest document" rule hurts factual accuracy too, just not as much. Sometimes, that rule ends up favoring a newer source that isn't as reliable over something older but officially recognized.

B. Error Analysis and Limitations

When we looked closely at the failure cases, we kept seeing the same three problems. First, some contradictions were so subtle you'd only catch them if you already knew something about the domain so the model missed them completely. Then, there were cases where two statements meant the same thing but were worded so differently, they ended up in different clusters, so the system never even compared them. Also, sometimes negations showed up in indirect ways like "no longer applicable" instead of just "not applicable" and the system's prompt missed these, too.

Now, about the evaluation set, it's pretty small, just 150 pairs, and everything comes from four specific domains. We didn't use a fancy automated method for the credibility weights in Table 3, those were just hand-tuned. As for the RFS and USR scores, there's subjectivity involved, even though we tried to balance it out by having two annotators and running a small blind user study.

VIII. Advantages

- Keeps contradiction handling and answer generation separate, so you can look at and test each part on its own something you don't get with the single-pass baselines in Table I.

- Utilizes direct, comprehensible heuristics rather than complex mathematics, thus simplifying the process for students to develop and debug.
- The agent setup is modular, so you can swap out and upgrade pieces (like using a better NLI prompt) without reworking the whole pipeline.
- Works with popular, free tools like LangGraph, FAISS/ChromaDB, and regular LLM APIs so you don't have to spend much.
- Offers you responses that are factually stronger and clearer compared to single-pass RAG approaches, and the delay is perfectly reasonable.

IX. Future Scope

- **Real-time enterprise integration:** plugging the pipeline into live data think ticketing systems or internal APIs instead of just pulling from a static index.
- **Adaptive credibility scoring:** replacing the hand-tuned weights with learned weights from user's feedback.
- **Human in the loop verification:** routing conflicts flagged as ambiguous by the Resolution Agent to a human reviewer for confirmation.
- **Domain-specific contradiction handling:** extending or finetuning the NLI-style classification prompt for specialized domains such as legal or medical text.
- **Larger-scale evaluation:** expanding the curated dataset beyond 150 pairs and running a larger, more statistically powered user study.

X. Conclusion

This paper lays out a straightforward multi-agent system that adds a contradiction detection and resolution layer right between retrieval and generation in RAG pipelines. Instead of piling everything onto one big model, it splits the work fact extraction, contradiction checks, credibility scoring, and resolution across distinct agents using LangGraph, each with a specific job. With this setup, the system consistently catches more contradictions and produces more accurate answers than Naive RAG, CoT RAG, or a simple Self-RAG baseline, and it doesn't slow things down enough to be a problem for internal, accuracy-focused projects.

There's no heavy math or complicated models here just clear, understandable heuristics. That means an undergrad at the end of their degree could actually design, build, and test this system in a single semester. Still, it leaves plenty of room for building out better, more trustworthy, multi-source RAG systems in the future.

XI. References

- [1] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," *NeurIPS*, vol. 33, pp. 9459–9474, 2020.
- [2] Z. Ji et al., "Survey of hallucination in natural language generation," *ACM Computing Surveys*, vol. 55, no. 12, 2023.
- [3] Y. Gao et al., "Retrieval-augmented generation for LLMs: A survey," *arXiv:2312.10997*, 2023.
- [4] T. Wu et al., "AutoGen: Enabling next-gen LLM applications via multi-agent conversation," *arXiv:2308.08155*, 2023.

- [5] K. Shuster et al., “Retrieval augmentation reduces hallucination in conversation,” Findings of ACL: EMNLP, 2021.
- [6] V. Karpukhin et al., “Dense Passage Retrieval for Open-Domain QA,” Proc. EMNLP, pp. 6769–6781, 2020.
- [7] J. Liu et al., “Lost in the Middle: How Language Models Use Long Contexts,” arXiv:2307.03172, 2023.
- [8] A. Asai et al., “Self-RAG: Learning to Retrieve, Generate, and Critique through SelfReflection,” arXiv:2310.11511, 2023.
- [9] Z. Jiang et al., “Active Retrieval Augmented Generation,” Proc. EMNLP, 2023.
- [10] A. Madaan et al., “Self-Refine: Iterative Refinement with Self-Feedback,” NeurIPS, 2023.
- [11] Y. Du et al., “Improving Factuality and Reasoning through Multiagent Debate,” arXiv:2305.14325, 2023.
- [12] S. Min et al., “FactScore: Fine-grained Atomic Evaluation of Factual Precision,” arXiv:2305.14251, 2023.
- [13] L. Wang et al., “Knowledge Graph Prompting for Multi Document Question Answering,” AAAI Conference on AI, 2023.
- [14] X. Chen et al., “Understanding and Mitigating Hallucinations in Large Language Models,” Nature Machine Intelligence, 2023.
- [15] W. Zhao et al., “A Survey of Large Language Models,” arXiv:2303.18223, 2023.