International Journal of Advanced Engineering Application (IJAEA)
Volume 3 Issue 11 Nov 2026
ISSN :3048-6807

AI-Powered Intelligent Research Assistant Using Large Language Models and Retrieval-Augmented Generation

[1] T. Sowjanya Kumari, [2]Chipurupalli Poojitha Ramya, [3]Basina Tejeswara Rao, [4] Bhavana Abhishith

[1] Assistant Professor, Department of Information Technology,
[2,3,4]B.Tech Student ,Department of Information Technology,
[1,2,3,4]GMR Institute of Technology, Rajam, Andhra Pradesh, India

Abstract

Finding trustworthy information online still means opening many pages, comparing them by hand and checking each claim against its source. Large language models can shorten this work, but when they answer from memory alone they sometimes state things that are not true and rarely say where an answer came from. This paper presents a research assistant that tries to close that gap by combining retrieval-augmented generation with a team of cooperating software agents. Given a question, the system plans the work, collects material from the web and from documents the user uploads, filters it for relevance and quality, and stores it as embeddings in a vector database. A Gemini model then drafts a report that rests only on the retrieved evidence. A separate reviewer agent checks the draft claim by claim and sends it back for revision until it passes. The finished report is delivered with its citations, either as a PDF file or in the web interface. The system is built with Python, FastAPI and React.js, and the agent workflow is controlled with LangGraph. In our testing the assistant produced organized, source-linked reports and reduced the manual effort of searching and summarizing.

Keywords: ***Large language models, retrieval-augmented generation, semantic search, multi-agent system, research assistant, hallucination.***

1. Introduction

Anyone who has tried to research a topic online knows how the process usually goes. A search returns pages of links, a few of them useful, and the real work starts afterwards: opening each page, skimming for the relevant part, comparing what different sites say, and deciding which of them to trust. The same question can lead to different answers depending on which pages happen to be read first. For a student writing a literature review or a professional preparing a briefing, this routine eats up hours that could have gone into thinking about the material instead of collecting it.

Large language models (LLMs) look like a natural fix, because they can summarize and explain in seconds. The trouble is that a model answering from its training alone knows nothing newer than its training data, and it can produce fluent statements that are simply wrong, a behavior usually called hallucination [1]. It also seldom tells the reader where a claim came from, so checking it means going back to manual search. Retrieval-augmented generation (RAG) addresses part of this problem. Instead of relying on memory, the model is handed passages that were retrieved for the question at hand and is asked to answer from them [2]. Even so, a systematic review of enterprise use found that standard retrieval dominates and that fewer than 15% of the studies dealt with real-time integration [3].

A single retrieval step does not settle the matter either. Noisy passages can still reach the model, and nothing checks the final text against them [4]. Recent work therefore splits the job among several cooperating agents, each responsible for one stage such as planning, retrieval or verification [5], [6]. We follow that line of thinking, but aim at a general-purpose tool rather than one limited to a single medical or scientific field.

In this paper we describe an assistant that accepts a research question, plans how to answer it, gathers evidence from the web and from the user's own documents, drafts a report tied to that evidence, and reviews the draft before releasing it. The agent workflow is managed with LangGraph, the interface is written in React.js, and the backend runs on Python with FastAPI. The tool is meant for students, researchers, educators and working professionals who need dependable, cited answers without doing every search by hand.

The work was guided by the following objectives:

to build an assistant that understands a user's question and answers it with an LLM;

to use RAG to pull reliable content from trusted websites and from local documents;

to divide the task among specialized agents for query understanding, retrieval, content checking and report writing;

to reduce hallucination by tying every answer to evidence that can be retrieved and verified;

to produce structured reports with proper citations, so that research becomes faster and easier to verify.

The rest of the paper is organized as follows. Section 2 reviews related work, Section 3 presents the proposed system, Section 4 gives implementation details, Section 5 discusses the results, and Section 6 concludes the paper.

2. Related Work

2.1 Multi-agent and retrieval-based systems in specialized domains

The base paper for this project, Queryome [5], answers biomedical research questions by orchestrating several agents that retrieve, reason over and synthesize findings from scientific papers. Its agents check the evidence repeatedly and attach citations, and the system was tested on several benchmarks. ER-MedRAG [4] attacks a related weakness, noisy retrieval, by training cooperating agents with reinforcement learning on top of hybrid retrieval; across six medical and reasoning datasets it gained roughly 3 to 6 percentage points over its baselines. In a similar spirit, Deng et al. [6] pair a planner agent with a retrieval agent that rewrites queries, judges relevance and reranks results, and they report gains of 5.3 to 6.1 points in Recall@10 and normalized discounted cumulative gain (NDCG) on the MIMIC-III and TREC-COVID collections.

Outside medicine, Aboukadri et al. [7] use RAG with embeddings and ranking to pull access-control policies out of agile user stories, reaching accuracies between 67.38% and 84.21% depending on the dataset. RefAI [8] applies GPT-4 Turbo to PubMed metadata for literature recommendation and summarization; 97% of the articles it returned were relevant and none of the papers was fabricated.

2.2 Automated literature review and report generation

Several groups have tried to automate the writing step itself. DORA AI Scientist [9] is a multi-agent team that works over more than 28 million scientific texts and user-supplied text to explore a topic, search for references and draft reports, with the aim of cutting manuscript preparation time and hallucination. InsightAgent [10] keeps a human in the loop for systematic reviews; tested on 15 biomedical reviews, it raised review quality by 27.2%, reached 79.7% of the quality of human-written reviews, and finished a review in about 1.5 hours. Wu et al. [11] built an LLM and RAG pipeline with several levels of quality control that reached human-level review quality on propane dehydrogenation catalysts while keeping the hallucination rate below 0.5% at 95% confidence. InteractiveSurvey [12] lets users steer the generation of survey papers, and produced a survey in around 35 minutes with a System Usability Scale (SUS) score of 84.4 out of 100.

2.3 Surveys, evaluation and hallucination

A few surveys frame the open problems. Hajar et al. [2] reviewed 97 RAG publications from 2019 to 2024 and point to retriever effectiveness, system integration and the lack of standardized evaluation as the main obstacles. Karakurt and Akbulut [3] examined 63 studies on enterprise use and found that 63.6% relied on GPT models, 80.5% used standard retrieval and fewer than 15% addressed real-time integration. On the evaluation side, Antal and Buza [13] benchmarked open-source LLMs inside a RAG pipeline using 227 thesis abstracts and 122 questions; retrieval and answer quality both varied with the type of question, which supports measuring the two parts separately as well as together. Lavrinovics et al. [1] review how knowledge graphs can reduce hallucination at the pretraining, inference and post-generation stages. Balakrishnan [14] looks at a different corner, using RAG, reinforcement learning from human feedback and fine-tuning to speed up data-pipeline generation.

Table 1 lists the systems from the paragraphs above that are closest to ours in purpose.

Table 1 Selected related systems

Ref.	Focus	Approach	Reported outcome
[5]	Biomedical question answering	Multi-agent RAG with repeated evidence checks	Evaluated on research-task benchmarks
[4]	Clinical reasoning	Multi-agent reinforcement learning, hybrid retrieval	About 3 to 6 points above baselines
[6]	Cardiology case retrieval	Planner and retrieval agents, reranking	Recall@10 and NDCG up 5.3 to 6.1 points
[7]	Policy extraction from user stories	RAG with embeddings and ranking	Accuracy 67.38% to 84.21%
[8]	Literature recommendation	GPT-4 Turbo with relevance ranking	97% relevant, no fabricated papers
[9]	Scientific report writing	Multi-agent team with LLMs and RAG	High writing quality; no numeric metric
[10]	Systematic reviews	Multi-agent framework with clustering	Quality up 27.2%; about 1.5 hours
[11]	Literature review generation	LLM and RAG with quality control	Hallucination below 0.5%
[12]	Survey paper generation	LLM with HyDE retrieval and RAG	About 35 minutes; SUS 84.4

2.4 The gap this work addresses

Reading these papers side by side shows a pattern. Many of the strongest results come from systems built around one field, most often biomedicine, and some authors say openly that this limits how far the results generalize [4], [5], [6]. Review-writing systems tend to work from one fixed corpus. Among the works we examined, few combine live web retrieval with the user's own documents, and fewer still place a dedicated reviewer in a revision loop that audits both the facts and the citations before anything reaches the user. The assistant described here is built to fill that space.

3. Proposed System

The assistant is organized as a chain of specialized agents that share one workflow. Figure 1 shows the whole path from the user's question to the final report. Each solid box is an agent or a data store, and the dashed boxes beneath it list what that part is responsible for.

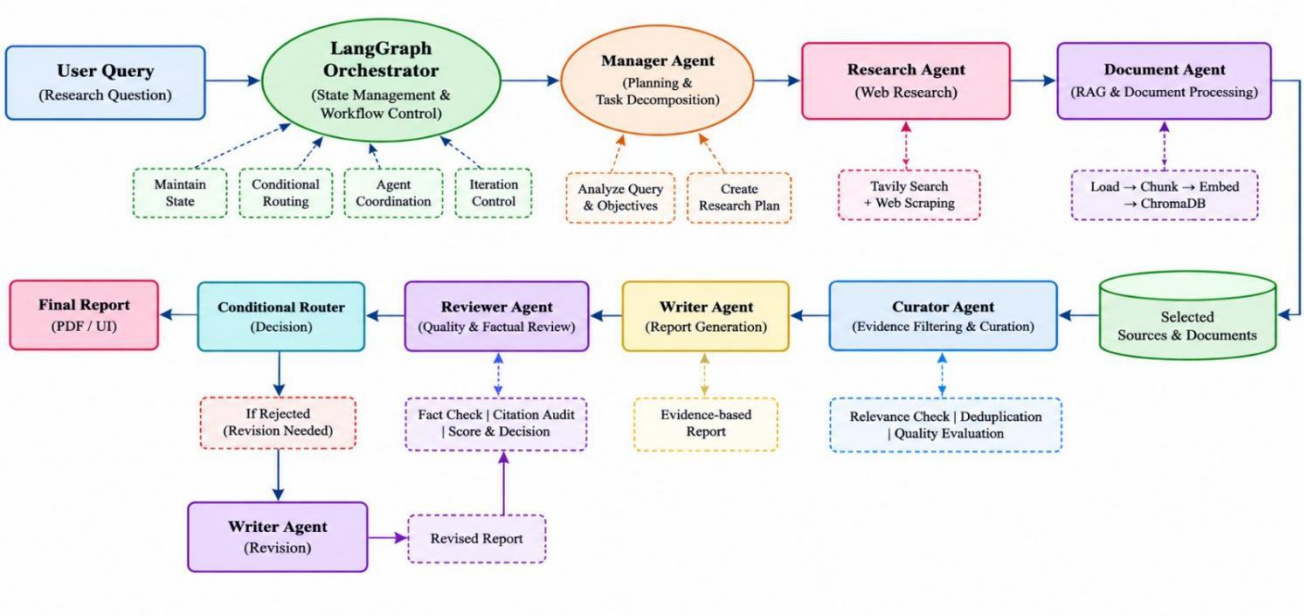


Fig. 1. Workflow of the proposed research assistant, from user query to final report.

3.1 Orchestration

Everything starts when the user types a research question. The query goes straight to a LangGraph orchestrator, which acts as the control center of the application. It keeps a shared state that records the query, the retrieved material, the current draft and the review status, so no agent has to guess what happened earlier. Through conditional routing it also decides which agent runs next, and it limits the number of repeat passes so that the revise-and-recheck cycle cannot run on forever.

3.2 Planning

The manager agent receives the query and works out what is really being asked. It splits the research goal into smaller tasks and writes a research plan that tells the later agents what to look for and where, whether on the web, in the uploaded files, or in both.

3.3 Information gathering

Two agents collect the material. The research agent searches the web with the Tavily search service and scrapes the pages it finds. The document agent handles files the user has uploaded: each file is loaded, split into chunks, converted to embeddings and stored in a ChromaDB vector database, so that passages can later be found by meaning rather than by exact keywords. The two agents work in parallel, and what they return is merged into one pool of selected sources and documents.

3.4 Curation and drafting

Not everything that was retrieved deserves a place in the report. The curator agent first checks each source against the query and drops those that do not match. It then removes duplicated content and rates the quality of what remains, keeping only reliable evidence. The writer agent receives this curated evidence and nothing else, and writes the first draft strictly from it. Keeping the writer away from unchecked material is one of the main ways the design tries to limit unsupported statements.

3.5 Review and self-correction

Before the user sees anything, the reviewer agent examines the draft. It runs a fact check, confirming that each claim agrees with the evidence it was written from, and a citation audit, confirming that sources are referenced correctly. It then gives the report a score and decides whether it is ready. A conditional router reads that decision. If the report is rejected, it goes back to the writer agent together with the issues that were flagged; the revised version returns to the reviewer, and the cycle repeats until the report is approved.

3.6 Delivery

Once approved, the report is locked as the final version and exported as a PDF file, or shown directly in the web interface. The citations and source references gathered earlier stay attached, so every part of the report can be traced back to the evidence behind it.

3.7 Retrieval-augmented generation pipeline

Underneath the document and research agents sits a standard RAG pipeline, shown in Fig. 2. The idea is to let the model answer from retrieved material instead of from whatever it happened to memorize during training. The system first collects text from documents or web sources and splits it into smaller chunks, because a short passage is much easier to match against a question than a whole file. Each chunk is then turned into an embedding, a list of numbers that captures its meaning, using the Gemini embedding model. The vectors are stored in ChromaDB together with the original text.



Fig. 2. Steps of the RAG pipeline, from document loading to answer generation.

When a user asks a question, it is embedded in the same way, and ChromaDB returns the chunks whose vectors lie closest to it. These chunks go to the Gemini LLM along with the question, and the model writes its answer from that context. Ahead of this step, source filtering and relevance checks remove material that does not fit the question, so weak text never reaches the model.

We use this arrangement to make the model lean on retrieved information rather than on its pretrained knowledge alone, which lowers the chance of incorrect or fabricated statements. It does not remove the problem. A model can still misread a good passage, and a poor retrieval can still slip through, which is why the reviewer agent described above stays in the loop.

4. Implementation

The backend is written in Python and exposes its functions through FastAPI. The agent workflow is defined as a LangGraph graph in which each agent is a node and the routing rules are conditional edges. Web retrieval uses Tavily search together with page scraping. Text chunks are converted to vectors with the Gemini embedding model and indexed in ChromaDB for semantic search, and a Gemini LLM writes the answers from the retrieved context. The front end is a React.js application where the user enters a question, uploads documents, and reads or downloads the finished report. Because retrieval, writing and review are separate nodes, any one of them can in principle be swapped, for example by changing the language model or the search source, without rewriting the rest.

5. Results and Discussion

5.1 Observed behavior

The assistant was developed end to end with Python, FastAPI and React.js, and it runs the full path described above: query understanding, retrieval from the web and from documents, curation, drafting, review and delivery. In use, it takes over the repetitive part of research, namely searching, reading and summarizing, and returns a structured report in which statements carry references to their sources. The reader can open those sources to confirm a claim, which is exactly what a plain chatbot answer does not allow.

Several design choices contribute to this behavior. Retrieval supplies current material that the language model alone would not have. The curator filters the pool before writing begins, so weak or repeated sources are less likely to shape the draft. The reviewer then compares the draft with its evidence, and the revision loop gives the system a chance to repair problems instead of passing them on. Together these steps are why we expect fewer unsupported claims than from a single-pass model, an expectation that is consistent with the gains other multi-agent RAG systems report [4], [6].

5.2 Choice of language model

The quality of the final report depends heavily on the model that writes it, so we compared several candidates. Fig. 3 shows the performance score each one reached, and Table 2 lists their context windows and intended uses.

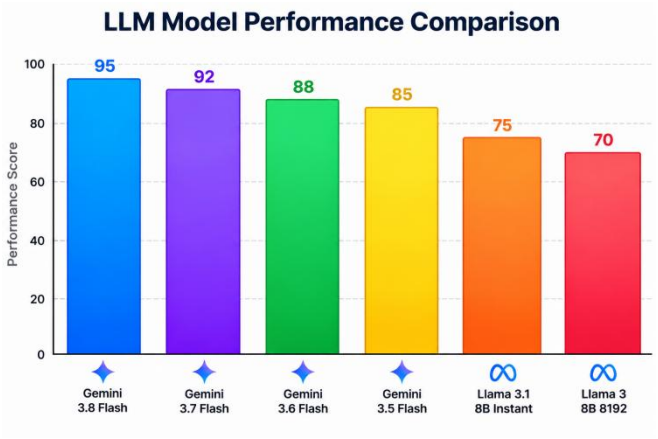


Fig. 3. Performance scores of the language models compared.

Two patterns stand out. First, the Gemini Flash models take the top four places with scores of 95, 92, 88 and 85, and each newer version gains three or four points over the one before it. Second, the two Llama models sit well below them at 75 and 70, a gap of ten points to the weakest Gemini model. Both Llama models are 8-billion parameter versions, so a lower score is not surprising, although Llama 3.1 8B Instant still finished five points ahead of the older Llama 3 8B.

The context windows in Table 2 point the same way. The Gemini Flash models accept about one million tokens, Llama 3.1 8B Instant 131 thousand and Llama 3 8B only 8 thousand. In a RAG system a larger window leaves room for more retrieved chunks and for the instructions the agents add, which matters when a report draws on many sources. The embedding model has a far smaller limit, 2,048 tokens per input, so chunks have to be cut small enough to fit it. Smaller models still have a place: Llama 3.1 8B Instant is listed for fast, light work such as classification, which could suit simple checks like relevance filtering.

Table 2 Models considered and their main characteristics

Model	Context window	Performance	Best use
Gemini 3.8 Flash	1M tokens*	Very high*	Complex agents, coding, research
Gemini 3.7 Flash	1M tokens*	Very high*	Reasoning, coding, tool use
Gemini 3.6 Flash	1M tokens*	High*	RAG, coding, summarization
Gemini 3.5 Flash	Verify model ID	Unverified	Verify provider support
Llama 3.1 8B Instant	131K tokens	Good for its size	Fast responses, classification
Llama 3 8B 8192	8K tokens	Moderate for its size	Simple chat, text processing
Gemini Embedding 001	2,048 tokens per input	MTEB: 68.17/100*	Embeddings and semantic search

* Approximate values; confirm against the provider's documentation. MTEB stands for Massive Text Embedding Benchmark.

Taken together, the comparison supports using a Gemini model for answer generation and the Gemini embedding model for retrieval. The scores should be read as a comparison between candidates, not as a measure of report accuracy; they say nothing yet about how often a finished report contains an unsupported claim.

5.3 Limitations

We should be clear about what has and has not been shown. This paper reports a working system, a comparison of candidate models and qualitative observations of its output. It does not yet include a controlled benchmark that measures hallucination rate, citation accuracy or retrieval quality against baselines. RAG reduces the chance of hallucination but cannot eliminate it. The quality of a report still depends on what the search service returns and on the language model that writes it, and the review loop adds time and cost, since each revision means another round of model calls. Work such as [13] suggests that retrieval and generation should be evaluated separately, and we plan to follow that approach.

6. Conclusion and Future Work

We presented a research assistant that combines RAG, an LLM and a team of specialized agents under a LangGraph orchestrator. A question is turned into a plan, evidence is gathered from the web and from uploaded files, the evidence is filtered, a report is drafted from it, and a reviewer checks the draft until it is accepted. The result is a structured, cited report delivered as a PDF file or in the interface, which spares the user much of the manual work of searching and cross-checking.

Future work will move in three directions. The first is a quantitative evaluation on standard benchmarks, scoring retrieval and generation separately [13]. The second is stronger grounding, for instance by adding knowledge graphs to catch hallucinations that survive the review [1]. The third is wider coverage: support for more document types and languages, and tighter real-time integration, which the literature identifies as a weak spot in current systems [3].

Acknowledgment

The authors thank Mrs. T. Sowjanya Kumari, Assistant Professor, for her guidance throughout this project, and GMR Institute of Technology for providing the facilities needed to carry it out.

References

[1] E. Lavrinovics et al., “Knowledge graphs, large language models, and hallucinations: An NLP perspective,” 2025.

[2] M. R. Hajar et al., “A systematic literature review of retrieval-augmented generation: Methods, applications, and future research directions,” 2025.

[3] E. Karakurt and A. Akbulut, “Retrieval-augmented generation (RAG) and large language models (LLMs) for enterprise knowledge management and document automation: A systematic literature review,” 2023.

[4] P. Shi, Y. Shui, R. Jin, et al., “ER-MedRAG: A multi-agent reinforcement learning framework for reliable clinical retrieval-augmented reasoning,” J. King Saud Univ. Comput. Inf. Sci., 2026.

[5] P. Punuru, N. Ibtehaz, S. Giri, et al., “Queryome: Orchestrating retrieval, reasoning, and synthesis across biomedical literature,” J. Biomed. Semantics, 2026.

[6] L. Deng, H. Hu, K. Lu, et al., “LLM-augmented multi-agent cooperative framework for medical case retrieval in cardiology,” J. King Saud Univ. Comput. Inf. Sci., 2025.

[7] S. Aboukadri, A. Ouaddah, A. Mezrioui, and I. El Asri, “Leveraging RAG and LLMs for access control policy extraction from user stories in agile software development,” IEEE Access, vol. 13, pp. 116462–116472, 2025, doi: 10.1109/ACCESS.2025.3586203.

[8] Li et al., “RefAI: A GPT-powered retrieval-augmented generative tool for biomedical literature recommendation and summarization,” J. Am. Med. Inform. Assoc., 2024.

[9] V. Naumov et al., “DORA AI Scientist: Multi-agent virtual research team for scientific exploration discovery and automated report generation,” 2023.

[10] R. Qiu et al., “Completing a systematic review in hours instead of months with interactive AI agents,” 2022.

[11] S. Wu et al., “Automated literature research and review-generation method based on large language models,” 2023.

[12] Z. Wen et al., “InteractiveSurvey: An LLM-based personalized and interactive survey paper generation system,” 2024.

[13] M. Antal and K. Buza, “Evaluating open-source LLMs in RAG systems: A benchmark on diploma theses abstracts using Ragas,” Acta Univ. Sapientiae, Informatica, 2025.

[14] A. Balakrishnan, “Enhancing data engineering efficiency with AI: Utilizing retrieval-augmented generation, reinforcement learning from human feedback, and fine-tuning techniques,” 2024.