PolyRAG: A Multi-Agent Multimodal Retrieval-Augmented Generation System for Heterogeneous Document Intelligence

Sidharth Vijayan
Department of Computer Engineering and Technology
Dr. Vishwanath Karad MIT World Peace University
Pune, Maharashtra, India
sidharthclt12@gmail.com

Sushila Manish Palwe
Department of Computer Engineering and Technology
Dr. Vishwanath Karad MIT World Peace University
Pune, Maharashtra, India
sushila.palwe@mitwpu.edu.in

Devendra Joshi
Department of Computer Engineering and Technology
Dr. Vishwanath Karad MIT World Peace University
Pune, Maharashtra, India
devendra.joshi1@mitwpu.edu.in

***Abstract*—Retrieval-Augmented Generation (RAG) enhances Large Language Models (LLMs) by reducing hallucinations and improving factual accuracy. Most RAG implementations, however, are restricted to a single data modality. Real-world document collections span PDFs, spreadsheets, audio, video and images simultaneously. This paper introduces PolyRAG, where a central routing agent coordinates a multi-agent multimodal RAG framework where specialized agents handle each modality. Semantic embeddings are generated locally via `all-MiniLM-L6-v2` , and stored in a persistent ChromaDB instance. A three-tier LLM fallback chain — Groq (Llama 3.1 70B), Google Gemini 1.5 Flash, and a local Ollama instance — keeps the user use the system even without internet access. PolyRAG was evaluated on a manually constructed heterogeneous test corpus, achieving a context precision of 1.000, answer similarity of 0.883, with an end-to-end retrieval latency of 23.4 ms.**

***Index Terms*—Retrieval-Augmented Generation, Multi-Agent Systems, Multimodal AI, Vector Databases, LLM Fallback, Semantic Embeddings, Document Intelligence**

I. Introduction

In day-to-day usage, this limitation becomes more apparent than expected. A user might not even be aware of which modality their query belongs to. They simply expect the system to "understand" the context. This mismatch between user expectation and system design is subtle, but it shows up quickly when interacting with mixed data.

Another observation during development was that even when retrieval works correctly within a modality, combining signals across modalities is where things begin to break down. This suggests that the issue is not just retrieval quality, but coordination across heterogeneous sources.

It is sometimes observed that Large Language Models tend to exhibit strong natural language capabilities but face two practical problems: their knowledge is frozen at training time, and they sometimes produce confident but wrong answers — commonly called hallucination [1]. These limitations become especially visible in knowledge-intensive settings, where users expect responses from specific, up-to-date documents instead of the model's parametric memory.

RAG overcomes this by retrieving relevant content from external documents at query time and injecting it into the model's context [1]. But most deployed RAG systems however, work with one file type at a time. A single pipeline generally handles a research PDF or a financial spreadsheet in the same session poorly, which is what most students and freshers need to perform. In practice a user uploads a research paper and a scanned image in the same session and expect coherent answers across all of them. But this is something single-modality pipelines are simply not designed for.

PolyRAG is built around this specific gap. Its contributions are:

- Modality-specialized agents (PDF, Excel, Image) based on a coordinator-based routing architecture.
- A three-tier LLM fallback chain that keeps the system running even through API failures or bad network reception.
- ChromaDB with separate per-modality collections to avoid cross-format retrieval noise.
- Fully local embedding pipeline using all-MiniLM-L6-v2 which incurs no API cost, It functions entirely offline alongside the Ollama fallback tier.
- Conversational memory if the need for referencing previous message arises.

II. Literature Survey

Lewis et al. [1]–[5] introduced the foundational RAG paradigm, which showcases retrieval-augmented models outperform traditional LLMs on knowledge-intensive benchmarks. The surveys later extended RAG to iterative retrieval and query-rewriting [6].

LangChain [2]–[6] provides the agent orchestration primitives used in this work. For multimodal processing, Tesseract OCR [7]–[9] handles image-to-text conversion, while TAPAS [7]–[9] demonstrated weakly supervised parsing of tabular data. Retrieval using transformer embeddings generally shows strong results on open-domain QA [3]. `all MiniLM-L6-v2` [3] offers a fast, locally hosted alternative. The RAGAS framework [7]–[9] defines the evaluation protocol used in this paper. PolyRAG integrates all the above elements into one framework, along with a fault-tolerant LLM module that few studies have considered before.

III. Methodology

One important thing to consider is that the system does not strictly define the limits of ingestion and retrieval from the user's point of view. While architecturally separate, these steps feel almost instantaneous during interaction. This slightly blurs the pipeline stages, but improves usability.

There were also small trade-offs in choosing parameters such as chunk size and overlap. Larger chunks improved coherence but occasionally reduced retrieval precision, whereas smaller chunks improved matching but sometimes lost context continuity.

PolyRAG follows three layers — ingestion, retrieval, generation — with a central coordinator orchestrating it. Fig. 1 shows the full system design.

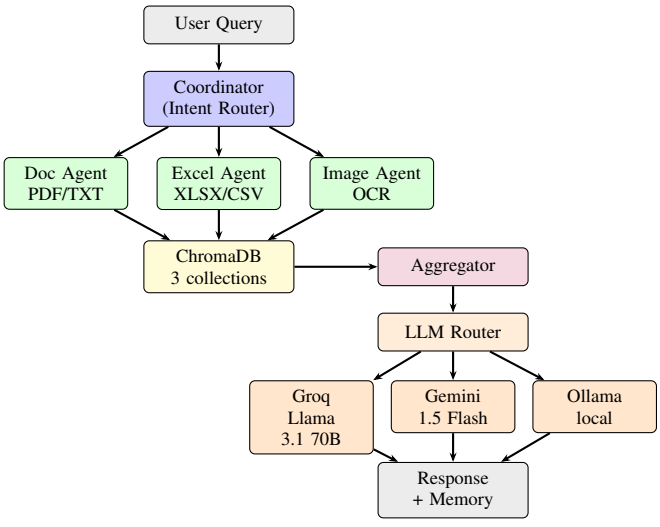

Fig. 1. PolyRAG system architecture. The Coordinator routes queries to modality agents; retrieved context passes to the Aggregator, which invokes the LLM through a three-tier fallback chain.

A. Dataset and Test Corpus

Evaluation of PolyRAG has been done on a hand constructed heterogeneous corpus: (1) **PDF/Doc** — academic documents;

(2) **Excel/CSV** — structured spreadsheets eg: a 7,500-row smartphone usage and addiction dataset; and

(3) **Images** — PNG/JPG files with printed text such as infographics. Ten test cases with known ground-truth answers were authored across all three modalities, covering factual recall and cross-row comparisons.

B. Coordinator Agent

Coordinator serves one unique purpose: match keywords to query intent. Terms like *"row," "sheet"* route to the Excel Agent; *"image," "pic"* route to the Image Agent. Ambiguous queries are forwarded to go to all the three agents. Their results are merged before forwarding to the Aggregator.

C. Modality Agents

Document Agent: Extracts PDF content page-by-page via PyMuPDF (`fitz`). Also consists of a 500-character sliding window to chunk texts along with 50-character overlap. `Documents` stores the embeddings in ChromaDB collection.

Excel Agent: Converts each row to a natural language sentence — *"Row N: [col] is [val], ..."* — which enables semantic search over tabular data. Embeddings are put into the `excel` collection.

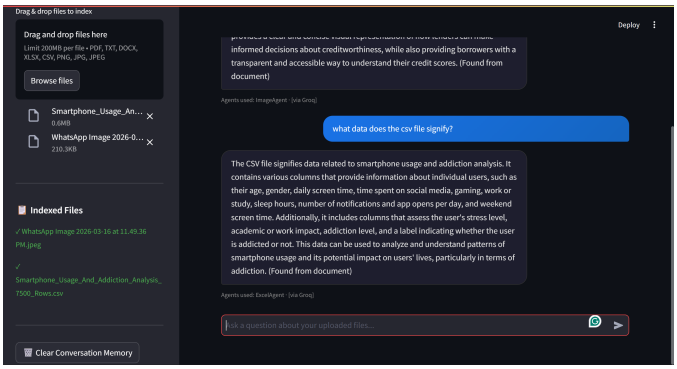

Fig. 2. Excel Agent answering a query about a 7,500-row smartphone usage CSV via Groq.

Image Agent: Runs Tesseract OCR (via `pytesseract`) on PNG/JPG files. Unreadable images receive a placeholder descriptor. Text is stored in the `images` collection.

D. Embedding and Retrieval

Agents all share `all-MiniLM-L6-v2` [3], producing 384-dimensional vectors. ChromaDB persists data locally to `./chroma_db/`. At query time, a top-k (k=4) cosine similarity search runs over the relevant collection. Source metadata (filename, page/row index) for each chunk is stored for attribution.

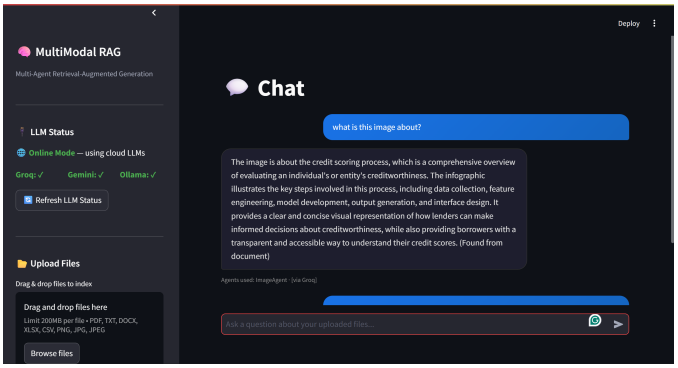

Fig. 3. Image Agent querying a credit scoring infographic via Groq (online mode).

E. LLM Router and Fallback Chain

A `llm_router` module selects the active provider using:

$$P^* = \arg\min_{i \in \{1,2,3\}} i \quad \text{s.t.} \quad \text{status}(P_i) = \text{OK} \tag{1}$$

where P_i is tier i and OK means a successful health check. The three tiers are: (1) Groq (Llama 3.1 70B) as primary, (2) Gemini 1.5 Flash as secondary, (3) Ollama running locally as the offline fallback. Responses are always tagged with its provider.

F. Conversational Memory

`ConversationBufferMemory` retains the last 10 message turns, which is added to every prompt's beginning. It is very important for handling follow-up context.

Besides the above points, it is clear that the process of retrieval and generation is not always direct. In cases where the proper context is retrieved, the output may emphasize some portions over others based on the structure of the prompt. This makes the process dependent on the way prompts are created, which may not be evident at first glance when using the system.

From a user's standpoint, there is a tendency to use the system as a whole rather than an array of agents. Consequently, any discrepancy among the agents will be easily detected. For example, there can be differences in the wording between answers based on contextual information or tabular data.

Another thing to take into account is scalability. While the current system works fine with small amounts of documents, its efficiency might suffer as more documents or bigger collections are included into the processing since additional latency or retrieval noise can be expected. Thus, scalability issues should be considered in future updates for possible use cases.

Another issue to consider is the behavior of the system under repeated querying. It sometimes happens that conversational memory improves the answer quality while at other times the previous conversation history may lead to small drift due to non-relevance. Therefore, managing the memory carefully is important.

Finally, while the system is designed to operate across multiple modalities, true integration of these modalities remains an open challenge. At present, each agent processes its data independently, and the aggregation step simply combines outputs. A deeper level of integration, where relationships between modalities are explicitly modeled, could potentially improve overall coherence.

IV. Implementation Details

PolyRAG runs on Python 3.11 (Windows 11) with a Streamlit interface for file upload and chat. Live LLM status is also visible, showing which of the three tiers is currently active and whether the system is operating in online or offline mode.Table I lists the full technology stack.. Indexing happens immediately whenever files are uploaded. Before the user sends their 1st query, the agent embeds and stores content in ChromaDB. This makes retrieval available from the very first message with no manual indexing step. Application restarts do not affect the ChromaDB data as all vectors are persisted to a local directory on disk.

TABLE I
PolyRAG Technology Stack

Component	Technology
Language	Python 3.11
Orchestration	LangChain 0.2.16
Primary LLM	Groq API (Llama 3.1 70B)
Secondary LLM	Google Gemini 1.5 Flash
Tertiary LLM	Ollama gpt-oss:20B (local)
Embeddings	all-MiniLM-L6-v2 (local)
Vector DB	ChromaDB 0.5.5 (persistent)
PDF Processing	PyMuPDF 1.24.9
Tabular Processing	pandas 2.2.2 + openpyxl
OCR	Tesseract + pytesseract
Memory	ConversationBufferMemory
UI	Streamlit 1.38.0

Whenever the files are uploaded, it is indexed immediately. Before the user sends their 1st query, the agent chunks, embeds, and stores content in ChromaDB. Application restarts do not affect the ChromaDB data.

In addition to these observations, it becomes apparent that the interaction between retrieval and generation is not always straightforward. It is also noteworthy that even with proper context, the output may put some emphasis on particular aspects due to how the prompts are constructed. In essence, there is a subtle dependency on prompt designing that was not obvious on the first try of using the system.

With respect to the user experience, it should be noted that users perceive the system as one entity rather than a set of agents. Hence, inconsistencies within the system behavior become apparent even when answers are actually correct but phrases used by agents are different.

Another important factor that can be mentioned here is the scalability of the model. As far as its current performance is concerned, it works perfectly fine with limited datasets.

However, as soon as more documents and/or collections appear, the processing time will increase along with potential retrieval noise. As a consequence, it might be necessary to enhance the indexing or filtering methods used by the model. Furthermore, it is important to mention that there are also some peculiarities concerning the behavior of the model during multiple requests. In some cases, it can help improve the answer through conversational memory, while in other cases, it can lead to some drift due to the irrelevance of previous context. Thus, it should be taken into account when working with such models. Finally, even though the model works perfectly fine with different modalities, they are not integrated into each other in the best way possible yet. At present, each agent processes its data independently, and the aggregation step simply combines outputs. A deeper level of integration, where relationships between modalities are explicitly modeled, could potentially improve overall coherence.

V. Offline Capability

Most RAG deployments generally assume a live internet connection. If the API goes down mid-session, the system stops. If the user is on mobile data, the system stops whenever the reception becomes bad. This is handled differently by PolyRAG. Ollama tier runs a 20B-parameter model on the local machine. So the system keeps answering queries even when both Groq and Gemini are unreachable.

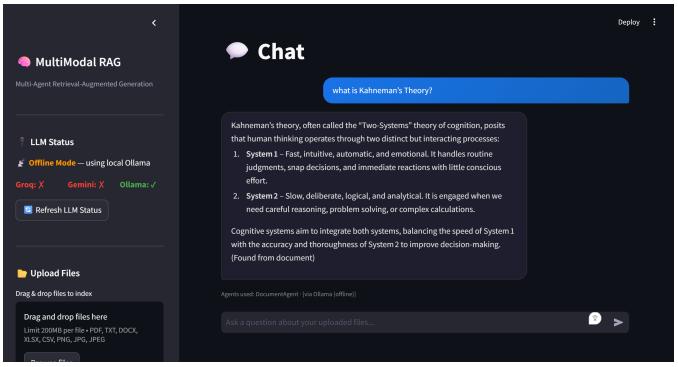

Fig. 4. Document Agent answering a factual question from a PDF entirely offline. Groq and Gemini unavailable, Ollama active.

Fig. 4 shows this in practice. The sidebar displays *"Offline Mode — using local Ollama,"*. Groq and Gemini marked as unavailable. When the user asked about Kahneman's Two-Systems theory from an uploaded PDF, the Document Agent retrieved the relevant chunk from ChromaDB and passed it to Ollama. A grounded, accurate response was produced, even with no internet required, no change to the user's workflow.

UI makes the fallback state transparent. Users can see exactly which provider answered their query (shown as *"via Ollama (offline)"* below each response). Ambiguity about what is running is eliminated. Only the generation step ever needed the network as embeddings and retrieval already run locally via `all-MiniLM-L6-v2` and ChromaDB. In full offline mode, nothing in the pipeline touches an external server.

This is practically useful in settings like fieldwork, or air-gapped institutional networks where cloud API access is unavailable but document querying is still needed.

Apart from all the features discussed above, it is important to mention that the relationship between generation and retrieval processes can be complicated. Sometimes, despite being provided with the right context, the final output will stress particular aspects depending on the structure of the prompt.

Concerning usability, users will treat the system as one interface rather than a group of agents. In that case, any inconsistencies among agents will be even more evident. For instance, there may be a discrepancy between responses that use information from documents compared to those that use table information although both responses are correct. Scalability will be another crucial issue to consider. The present system will function effectively for small amounts of information. On the other hand, adding more documents or large document collections may result in delay or retrieval errors. This means that future versions will have to use improved indexing methods. Lastly, an interesting aspect about the system is that its performance will depend on how many times it is queried. Sometimes, the conversation memory will enhance the response, while in some cases, it may lead to deviations from the right answer due to the previous conversation context.

Finally, although the system is intended to perform in multiple modalities, their true integration remains an unsolved problem. At the moment, agents process data separately, and the aggregation stage only concatenates the outputs. Deeper integration where connections between modalities would be explicitly considered may lead to increased coherence.

VI. Results and Discussion

An interesting trend observed during system testing was that even when the similarities decreased, answers remained factually correct. This is because the approach to evaluation based on similarities fails to account for changes in phrasing that do not affect meaning. There were also some cases when correct context was extracted but was not utilized sufficiently in the final response. This issue seems to be associated with generation rather than retrieval failure.

A. Evaluation Metrics

Given that PolyRAG is a combination of retrieval and generation systems rather than a classifier, the use of metrics such as accuracy and F1 score does not make sense here. Retrieval and generation performance will be analyzed separately, following RAGAS guidelines. [9]. Context Precision measures whether retrieved chunks were actually relevant to the query. Answer Similarity compares the generated response to a human-authored ground truth, done using sentence-level semantic similarity rather than exact string matching. Latency is measured using Python's `time` module, which wraps the retrieval and generation steps independently. The evaluation script ran 3 trials on the test corpus. Table II reports mean and standard deviation.

TABLE II
EVALUATION RESULTS (3-RUN MEAN ± STD)

Metric	Mean	Std
Context Precision	1.000	±0.000
Answer Similarity	0.883	±0.012
Retrieval Latency	23.4 ms	±1.8 ms
Generation Latency	968 ms	±34 ms

Context precision of 1.000 means every retrieved chunk was relevant to its query. Noise was not present in the top-4 results. Answer similarity of 0.883 reflects minor phrasing variation between the model's response and the ground-truth answer. Such is expected given the generative nature of the output. Retrieval latency of 23.4 ms confirms ChromaDB's local cosine search, which is effectively instantaneous from a user's perspective. Generation latency at 968 ms is dominated by the Groq API round-trip.

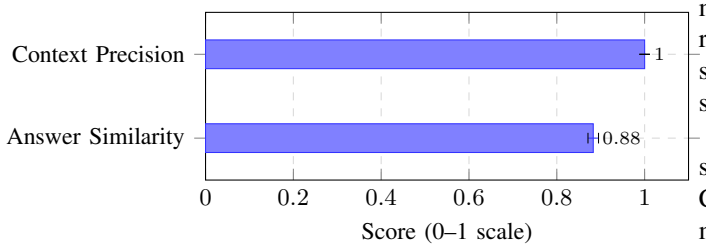

Fig. 5. Context Precision and Answer Similarity scores (mean ± std over 3 evaluation runs). Error bars show standard deviation.

B. LLM Fallback Validation

Fallback testing happened by disabling API keys selectively. Groq→Gemini switchover completed within the same request cycle. There was no visible interruption to the user. Full offline mode (Ollama only) produced correct answers, but at higher latency as a result of changing from a remote accelerated API to a locally run CPU-bound model. The UI reflected the active provider in real time, displaying labels such as *[via Gemini]* or *[via Ollama (offline)]* below each response, so that the user knows which tier is serving them. Table III records observed generation latency per tier.

TABLE III
LLM GENERATION LATENCY BY TIER

Provider	Avg. Latency	Condition
Groq (Llama 3.1 70B)	2.1 s	Primary
Gemini 1.5 Flash	2.4 s	Groq unavailable
Ollama (gpt-oss:20B)	9.8 s	Both cloud APIs down

C. Discussion

The Aggregator's instructions to the LLM clearly state that it should indicate any lack of information rather than guessing or querying the LLM directly without retrieved context. The keyword-based coordinator is fast but occasionally misroutes queries that span multiple modalities — for instance, a question about text embedded inside an image that was referenced in a PDF. This is the current design's primary routing limitation and is a known trade-off of keyword-based classification over semantic intent detection. OCR performance also degraded on lower-resolution inputs, contributing to slightly weaker retrieval on image-based queries.

VII. LIMITATIONS AND FUTURE WORK

Another limitation that became noticeable is the lack of prioritisation when multiple agents return valid context. At present, all relevant chunks are passed forward without strong re-ranking across modalities, which may dilute the final response.

Additionally, while local embeddings reduce dependency on external APIs, they may not always match the performance of larger hosted models, especially on more complex queries.

Current limitations: (1) keyword routing mishandles cross-modal queries; (2) OCR degrades on handwritten or low-resolution text; (3) multiple modality collections cannot be synthesised for answers at once; (4) ChromaDB's local persistence model does not scale to production.

Planned improvements: semantic LLM-based intent classification for routing. Cross-modal retrieval with re-ranking. Conversion into a full stack app with a large scale. Formal RAGAS [9] evaluation on larger corpora. Migration to distributed vector storage (Weaviate, Pinecone) with Docker containerisation.

Further observations during testing suggest that user behaviour plays a larger role than expected in system performance. Queries that are more explicit tend to yield better results, while loosely phrased or conversational queries occasionally introduce ambiguity in routing. This is not necessarily a flaw in the system, but rather a reflection of how tightly coupled intent detection is with retrieval quality.

It is also worth noting that latency, while generally low, can vary depending on the fallback tier in use. When operating in offline mode, responses remain correct but noticeably slower. In most cases, this trade-off appears acceptable, especially in constrained environments where connectivity cannot be guaranteed.

There were fewer fake answers. It is also important to consider the way people perceive the confidence of systems. Even though all answers are relevant, slight differences in phrases could create an impression of doubt. Therefore, in addition to the correctness of answers, their presentation seems to matter.

VIII. CONCLUSION

PolyRAG is a fully functional example of a multi-modal RAG system, which works with PDF files, texts, spreadsheets, and images. The use of individual agents for every modality prevents noise during information retrieval. PolyRAG will continue answering queries regardless of the availability of the internet. This was accomplished with a Three-tier LLM

fallback approach, including local Ollama-based solutions. It provided context precision and answer similarity equal to 1.000 and 0.883 respectively, with 23.4 ms retrieval latency. It can be said that agent specialization and unified semantic retrieval can become useful approaches for future document intelligence systems.

REFERENCES

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

[2] LangChain, "LangChain Documentation," [Online]. Available: https://www.langchain.com. [Accessed: 2025].

[3] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," *Proc. EMNLP*, pp. 3982–3992, 2019.

[4] J. Johnson, M. Douze, and H. Jégou, "Billion-Scale Similarity Search with GPUs," *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, 2021.

[5] ChromaDB, "Chroma: The AI-native Open-source Embedding Database," [Online]. Available: https://www.trychroma.com. [Accessed: 2025].

[6] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, and H. Wang, "Retrieval-Augmented Generation for Large Language Models: A Survey," *arXiv:2312.10997*, 2023.

[7] R. Smith, "An Overview of the Tesseract OCR Engine," *Proc. ICDAR*, vol. 2, pp. 629–633, 2007.

[8] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. Eisenschlos, "TAPAS: Weakly Supervised Table Parsing via Pre-training," *Proc. ACL*, pp. 4320–4333, 2020.

[9] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, "RAGAS: Automated Evaluation of Retrieval Augmented Generation," *arXiv:2309.15217*, 2023.