WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

Multi-Model RAG: An Agentic Retrieval-Augmented Generation System with Hybrid Retrieval and Grounded Answer Verification

Mr. Nagaraj A, Harsh Kumar, Kanav Mankotia, Daksh Sharma, Ishan Farooq

Jyothy Institute of Technology, Bangalore, India Professor of Practice, Dept. of CSE
nagaraj.a@jyothyit.ac.in, 1jt23cs054@jyothyit.ac.in, 1jt23cs069@jyothyit.ac.in, 1jt23cs039@jyothyit.ac.in, 1jt23cs065@jyothyit.ac.in

ABSTRACT—

Large language models answer fluently but are not grounded in a user's private or current information, and basic retrieval-augmented generation (RAG) pipelines rely on a single retriever, cannot choose between knowledge sources, rarely check whether an answer is supported, and become unsafe when retrieved content can trigger actions. This paper presents Multi-Model RAG, an agentic RAG system delivered to users as the Praxis workspace. A Next.js interface and a FastAPI backend stream answers and live pipeline stages over Server-Sent Events. A three-level router of deterministic rules, a cross-encoder knowledge-base probe and an LLM classifier selects one of eleven tools, including knowledge-base retrieval, multi-tier web search, image analysis and approval-gated email, Telegram and file actions. Knowledge-base retrieval fuses FAISS dense search and BM25 by reciprocal rank fusion, reranks with a cross-encoder, and declines when no passage clears a relevance floor. Answers are generated by local Qwen2.5 or Llama 3.2 models through Ollama, or by Gemini and Claude, and a verification module flags unsupported numbers, identifiers and names. On a 64-question evaluation, hybrid retrieval raised recall@5 from 0.781 to 0.938, retrieval-first routing raised end-to-end answer correctness from 0.094 to 0.672, all 24 unanswerable questions were declined, every fabricated and number-swapped answer in the verifier test set was caught, and none of 21 prompt-injection attacks produced a dispatchable message. The system runs on a 4 GB laptop GPU and offers a transparent, verifiable assistant for document-centric question answering.

Keywords— Retrieval-Augmented Generation, Large Language Models, Hybrid Retrieval, Cross-Encoder Reranking, Agentic Routing, Answer Verification, Prompt Injection, Local LLM Deployment.

1. INTRODUCTION

Large language models (LLMs) such as Llama, Qwen, Gemini and Claude can answer questions, summarise text and write code with remarkable fluency, yet on their own they know only what they saw during training. They cannot read an organisation's private documents, are unaware of events after their training cut-off, and, when uncertain, often produce a confident but invented answer. Retrieval-augmented generation (RAG) addresses this by retrieving relevant passages from a knowledge source and conditioning generation on them, which grounds the answer in updatable evidence [1]. For students, researchers and organisations who need answers drawn from their own reports, notes and current web information, grounding is a prerequisite for trust.

Basic RAG systems nevertheless have well-known weaknesses. A single dense vector search [2] misses passages whose wording differs from the question, while lexical ranking such as BM25 [3] cannot handle paraphrase; rank fusion [4] can combine the two. Sentence encoders [5] and cross-encoder rerankers [6] sharpen relevance, and vector libraries such as FAISS [7] make dense search fast on local hardware. Even so, retrieved passages are often noisy, the model may ignore the evidence or cite a source for a claim the source never makes, and the system has no notion of when to use the user's documents, when to search the web and when simply to converse. Agentic frameworks such as ReAct let the model choose tools [8], and Self-RAG teaches a model to critique its own retrieval and output [9], but the first depends on the model's judgement, which is unstable for small local models, and the second requires a specially trained model. Evaluation frameworks such as RAGAS rely on an LLM judge [10], and once an assistant can send messages or write files, instructions hidden in retrieved content can hijack it [11]. Table I summarises these foundations and the limitation of each that this work addresses.

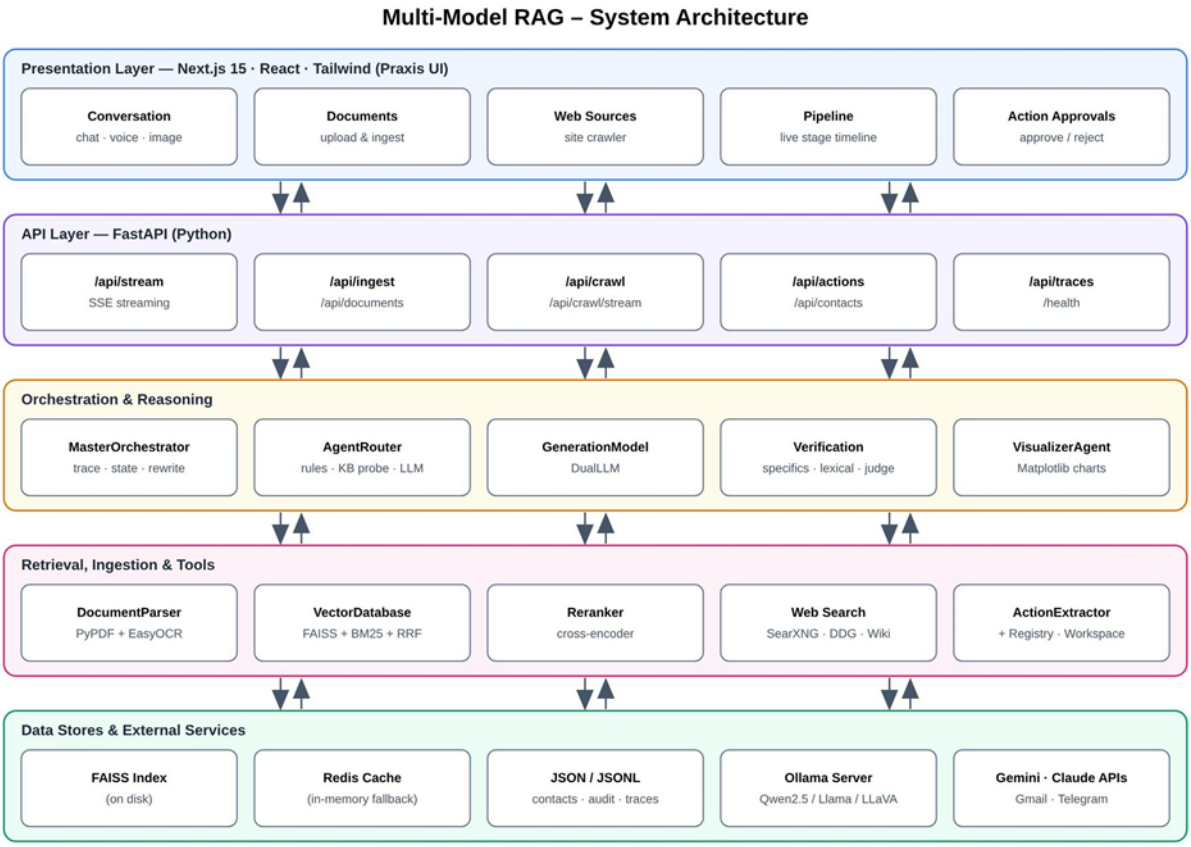


Fig. 1. System architecture of Multi-Model RAG

TABLE I. Summary of Foundational Work and Limitations Addressed

Ref.	Approach	Limitation addressed in this work
[1]	Retrieval-augmented generation	Single retriever; no answer verification or abstention
[2]	Dense passage retrieval	Misses rare and exact terms found by lexical search
[3]	BM25 lexical ranking	No understanding of synonyms or paraphrase
[4]	Reciprocal rank fusion	Ignores score magnitude, so it cannot express confidence
[5]	Sentence-BERT encoders	General-purpose models, not tuned to a domain
[6]	BERT cross-encoder reranking	One pass per query–passage pair; small pools only
[7]	FAISS vector search	Vector search only; lexical search left to the application
[8]	ReAct tool use	Tool choice left to the model; unstable with small models
[9]	Self-RAG	Requires a specially trained model
[10]	RAGAS evaluation	LLM judge found lenient with a small model
[11]	Indirect prompt injection	Mitigations largely left open

These capabilities are usually studied in isolation. This paper presents Multi-Model RAG, delivered to users as the Praxis agentic workspace, which combines them in one local-first system that decides how each question should be answered, retrieves accurately, streams transparent answers, declines when evidence is missing and acts only with human approval. The key contributions of this paper are as follows:

A three-level agentic router that applies deterministic rules, then a cross-encoder knowledge-base probe, and only then an LLM classifier to select one of eleven tools.

- A hybrid retrieval stack that fuses FAISS dense search and BM25 by reciprocal rank fusion, reranks with a cross-encoder and abstains below a relevance floor.
- A fast verification module that names unsupported numbers, identifiers and proper nouns, running in parallel with automatic chart generation.
- An action layer in which recipients come only from a contact allowlist, every action is approved by the user and every attempt is audited.
- An evaluation harness and 349 automated tests measuring routing, retrieval, answer correctness, verification, abstention, prompt-injection resistance and latency.

Section II describes the proposed methodology and its implementation, Section III presents the implementation screenshots and evaluation results, Section IV outlines the future scope and Section V concludes the paper.

PROPOSED METHODOLOGY

This section describes how Multi-Model RAG turns a user's question into a grounded, verified answer. Three design rules guide every component. First, deterministic logic precedes probabilistic model calls: routing rules, the knowledge-base probe, recipient resolution and chart figures are computed in code, and the LLM is used only where judgement is genuinely needed. Second, degraded results are labelled rather than hidden. Third, every outbound action remains a draft until a human approves it.

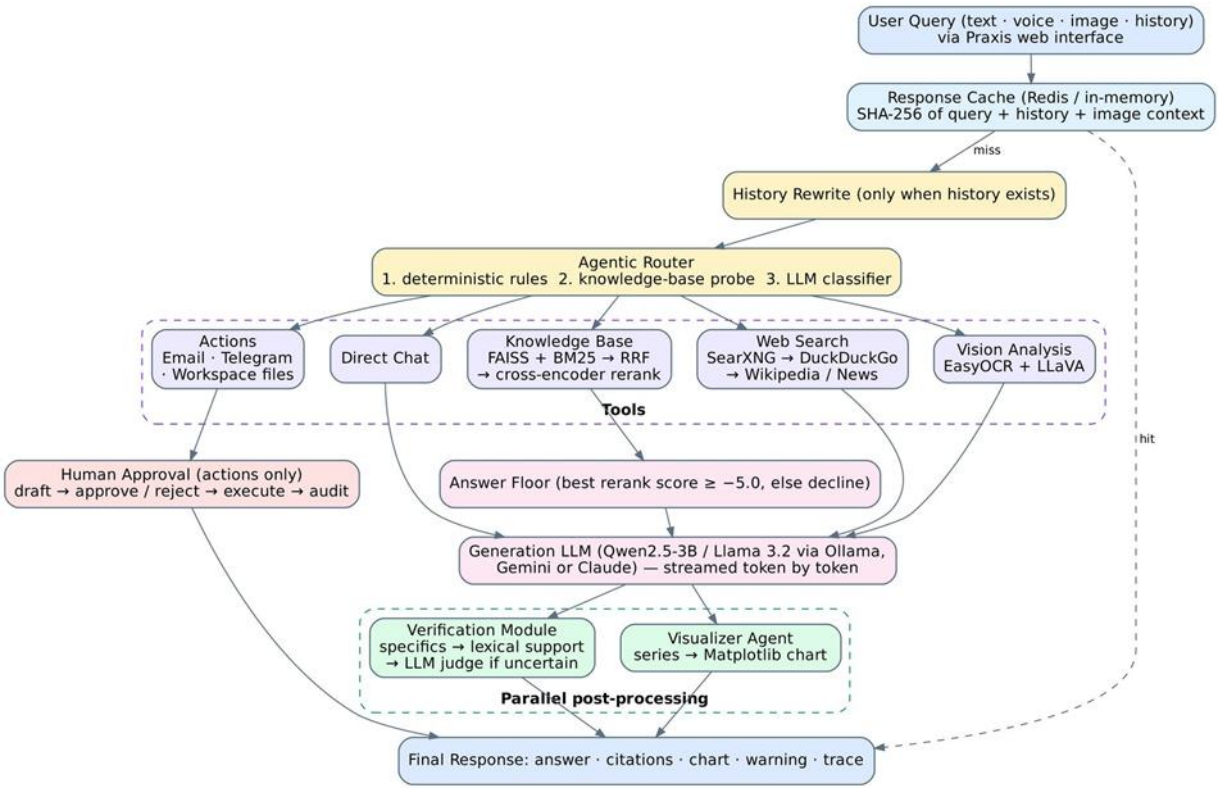


Fig. 2. Request-processing workflow from user query to final response

A. System Architecture

As illustrated in Fig. 1, the system is organised into five layers so that each component can be developed, tested and replaced independently. The *presentation layer* is the Praxis interface built with Next.js 15, React and Tailwind CSS, with Conversation, Documents, Web Sources, Pipeline and action-approval views. The *API layer* is a FastAPI service exposing /api/stream for streamed answers (GET for text, POST with images), /api/ingest and /api/documents,/api/crawl and /api/crawl/stream, /api/actions and /api/contacts, /api/traces and /health.

The *orchestration and reasoning layer* contains the MasterOrchestrator, the AgentRouter, the GenerationModel, the VerificationModule and the VisualizerAgent. The *retrieval, ingestion and tools layer* contains the DocumentParser with OCR fallback, the VectorDatabase (FAISS, BM25 and rank fusion), the cross-encoder reranker, multi-tier web search and the action components. The *data and external services layer* holds the FAISS index on disk, a Redis cache, JSON/JSONL stores for contacts, audit entries and traces, the local Ollama server and external APIs. Fig. 2 shows how a single request flows through these layers. All services except Ollama run under Docker Compose; Ollama runs on the host so that it can use the GPU.

B. Data Ingestion and Preprocessing

The quality of a RAG answer is bounded by the quality of the indexed text. The embedded text layer of a PDF cannot be trusted: scanner applications often write garbage into it, and image-only scans have none. The DocumentParser therefore extracts each page with PyPDF and scores its text quality as the fraction of its words that are whole words in the embedding model's vocabulary V:

$$Q(\text{page}) = |\{w \in W : w \in V\}| \,/\, |W| \qquad (1)$$

where W is the set of words on the page. Measured on real data, OCR garbage scores 0.00–0.30 and normal prose 0.84–0.91, so pages with $Q < 0.55$ or fewer than 12 words are treated as missing, rendered at 170 DPI with pypdfium2 and re-read with EasyOCR. The text is split with LangChain's recursive character splitter into chunks of about 1,000 characters with 200 characters of overlap, each tagged with its source file, page number and extraction method. Chunks are embedded with all-MiniLM-L6-v2 [5] into 384-dimensional vectors and stored in a persistent FAISS index [7], and a dependency-free BM25 index is rebuilt lazily over the same chunks. A BeautifulSoup crawler adds single web pages or site maps of up to 15 pages, streaming its progress to the interface. At query time, follow-up questions are rewritten into standalone form using the chat history, but only when history exists, which avoids an unnecessary model call.

C. Agentic Routing and Orchestration

For every request the MasterOrchestrator first checks a response cache keyed by the SHA-256 hash of the query, history and image context; entries are held in Redis for one hour, with an automatic in-memory fallback. On a miss it rewrites the query, routes it, executes the selected tool and records a timed trace with flags for cached, degraded and abstained responses and for the verification verdict. Routing matters: in an early version, a router that treated reader-style questions as general knowledge sent 57 of 64 document questions to web search, dropping end-to-end correctness to 9%. The AgentRouter therefore uses the three levels of Algorithm 1 to choose among eleven tools:

Search_Knowledge_Base, Web_Search, Direct_Chat, Visualize_Data, Vision_Analysis, Send_Email, Send_Telegram, Read_Email, Workspace_Task, Ambiguous_Query and Send_WhatsApp, which is currently mapped to Telegram

Algorithm 1: Three-Level Agentic Routing Input: standalone query q

Output: selected tool

1: **if** q matches an ordered deterministic rule **then**

2: **return** the rule's tool ▷ sends before reads and file writes 3: **if** q does not ask for current information **then**

4: $s \leftarrow \max_p$ CrossEncoder(q, p) ▷ knowledge-base probe

5: **if** $s \geq -3.0$ **then return** Search_Knowledge_Base

6: $t \leftarrow$ LLMClassifier(q, eleven tools)

7: **if** t cannot be parsed **then return** Search_Knowledge_Base 8: **return** t

Level 1 resolves unmistakable intents, such as "email Ali the meeting notes", without calling a model. Level 2 probes the knowledge base unless the query asks for something current, such as the latest news, today's price or the weather. Level 3 is used only when neither applies, and any unparseable classifier output falls back to the read-only knowledge-base tool, so that a routing error can never trigger an action.

D. Hybrid Retrieval and Reranking

Dense embeddings capture paraphrase but miss short bulleted facts with distinctive wording, which lexical search finds easily. The VectorDatabase therefore retrieves candidate pools of 20 passages from FAISS and from BM25 (Fig. 3). BM25 scores a chunk d for query q as [3]

$$\text{score}(q,d) = \Sigma_{t \in q} \text{IDF}(t) \cdot \text{TF}(t,d) \quad (2)\ \text{TF}(t,d) = f(t,d)(k_1+1) \,/\, [f(t,d) + k_1 B(d)] \qquad (3)$$

where $f(t,d)$ is the frequency of term t in d, $B(d) = 1 - b + b \cdot |d|/\text{avgdl}$ normalises for chunk length, and the implementation uses $k_1 = 1.5$ and $b = 0.75$. The two ranked lists are combined by reciprocal rank fusion [4], which needs no score calibration:

$$\text{RRF}(d) = \Sigma_{r \in \{\text{dense, BM25}\}} \, 1 \,/\, (k + \text{rank}_r(d)),\ k = 60 \qquad (4)$$

The fused top-10 candidates are scored jointly with the query by the ms-marco-MiniLM-L-6-v2 cross-encoder [6], [5], and the five highest-scoring passages P_5 form the context. An answer is generated only if

$$\max_{p \in P} 5 \; s(q,p) \geq \tau_a,\ \tau_a = -5.0 \qquad (5)$$

otherwise the system replies that the answer is not in the user's documents instead of guessing. The floor is skipped when the question carries its own data or an image, and the same cross-encoder powers the router's knowledge-base probe with the stricter threshold of −3.0.

E. Multi-Model Generation, Web Search and Vision

A DualLLM provider serves several models behind one streaming interface, selectable per conversation: Qwen2.5-3B [12] (the default) and Llama 3.2 run locally through Ollama, and Gemini 2.0 Flash and Claude are available through API keys. Prompts number the retrieved sources so that the answer can cite them inline. Two configuration changes came from measured failures: the Ollama context window was raised to 8,192 tokens after long web contexts were truncated at 4,096, and the keep-alive was set to 30 minutes to avoid a model reload of about 9 s after idle periods.

The web path builds the query (the raw query first, then per-period queries for time-series questions) and searches a self-hosted SearXNG instance, then DuckDuckGo, then Wikipedia and Google News. Result pages and PDFs are fetched, deduplicated and merged up to 16,000 characters, and a rate-limited search is labelled as degraded and never cached. For images, the ImageAnalyzer combines EasyOCR text with a description from the LLaVA vision model [13].

F. Answer Verification and Visualization

After generation, the VerificationModule and the VisualizerAgent run in parallel. Verification (Algorithm 2) relies first on deterministic checks: a lexical support score measuring how much of the answer is found in the retrieved sources, and an unsupported-specifics check that extracts numbers, identifiers and proper nouns from the answer and reports those absent from every source. An LLM judge is consulted only in the uncertain middle band.

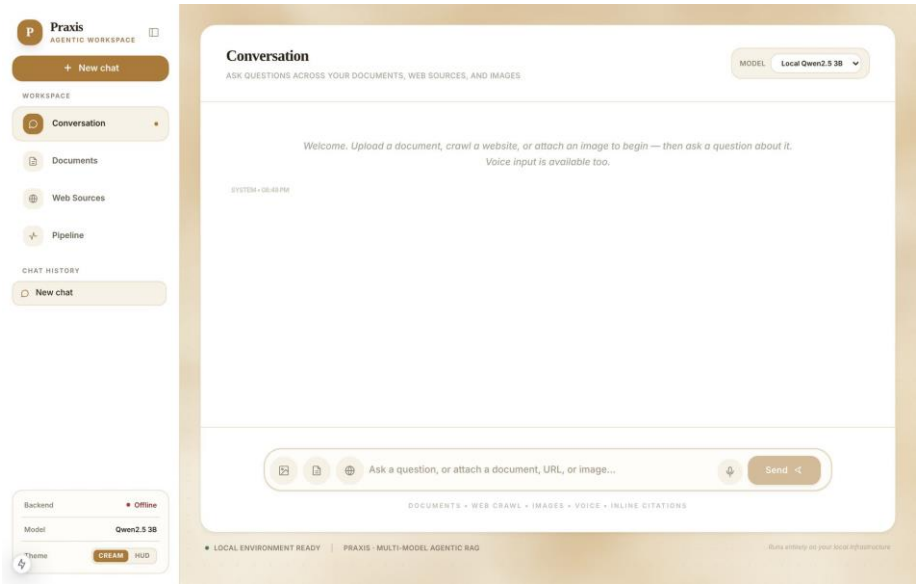


Fig. 4. Conversation view (Cream theme)

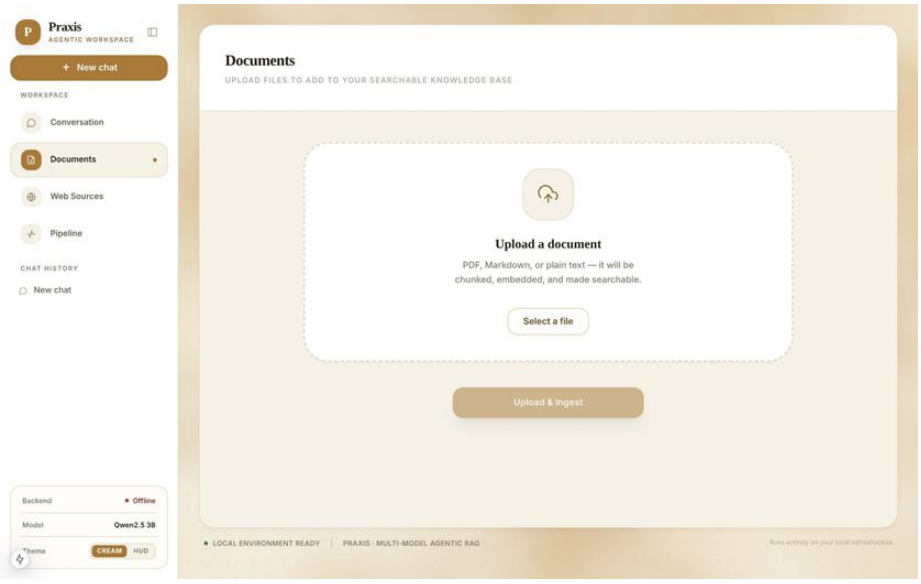


Fig. 5. Documents view: upload and ingest

Algorithm 2: Fast Answer Verification

Input: answer a, retrieved sources S

Output: verdict and unsupported tokens U

1: $\ell \leftarrow$ LexicalSupport(a, S)

2: **if** $\ell \leq 0.30$ **then return** FAIL

3: $U \leftarrow$ specifics of a (numbers, IDs, names) absent from S

4: **if** U is not empty **then return** FAIL, U

5: **if** $\ell \geq 0.75$ **then return** PASS

6: **return** LLMJudge(a, three trimmed chunks of S)

Checking specifics before the automatic pass is deliberate: a small model can copy most of a source and still change one number, which lexical overlap alone accepts. A failed answer is shown with a warning naming the unsupported tokens, and only verified, non-degraded answers are cached.

The VisualizerAgent extracts numeric series from the answer or search results and renders charts with Matplotlib using a fixed renderer, so chart values are computed in code rather than by the model.

G. Safe Actions and Security

Because the assistant reads untrusted documents and web pages, it follows the indirect prompt-injection threat model [11], and its safeguards are enforced in code rather than in prompts. For email, Telegram and workspace-file actions the model returns only a recipient name, and the code resolves the address from the saved contacts store; ambiguous names resolve to nothing, and allowlist membership is checked again at dispatch. Each action is shown as a draft card with Approve and Reject buttons, file paths are rejected if they resolve outside the praxis-workspace folder, and every drafted, approved, rejected, sent or blocked action is written to an audit log. Gmail (send and read) and the Telegram Bot API are integrated; WhatsApp through Twilio is available but disabled by default, and secrets are supplied through environment variables. Data leaves the machine only when web search or an API model is chosen.

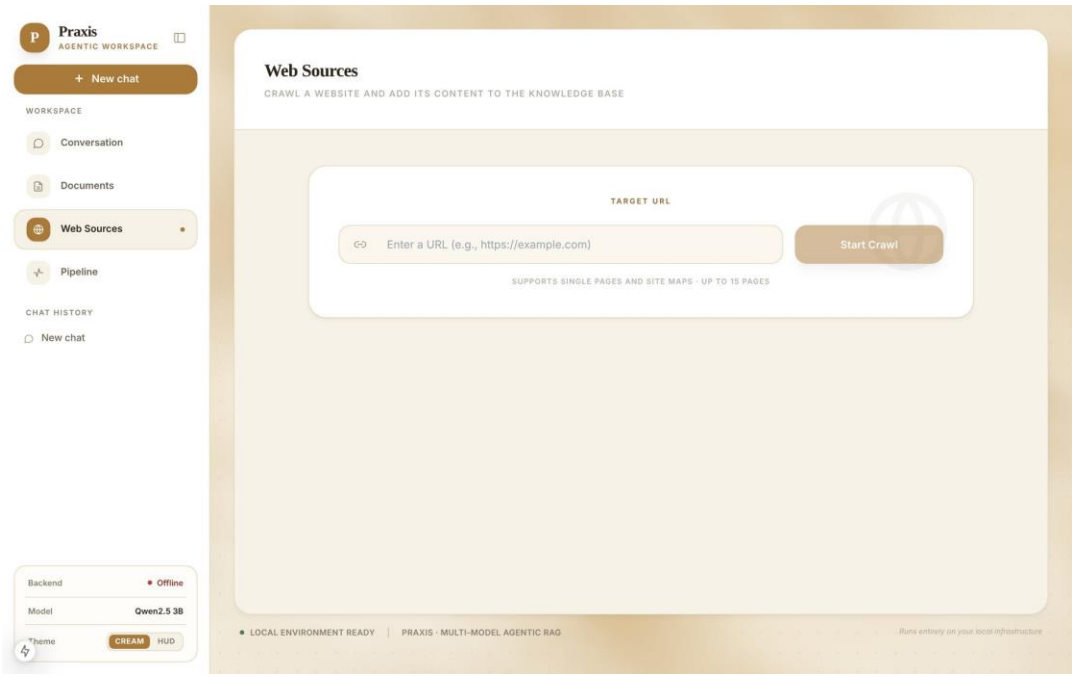


Fig. 6. Web Sources view: website crawler

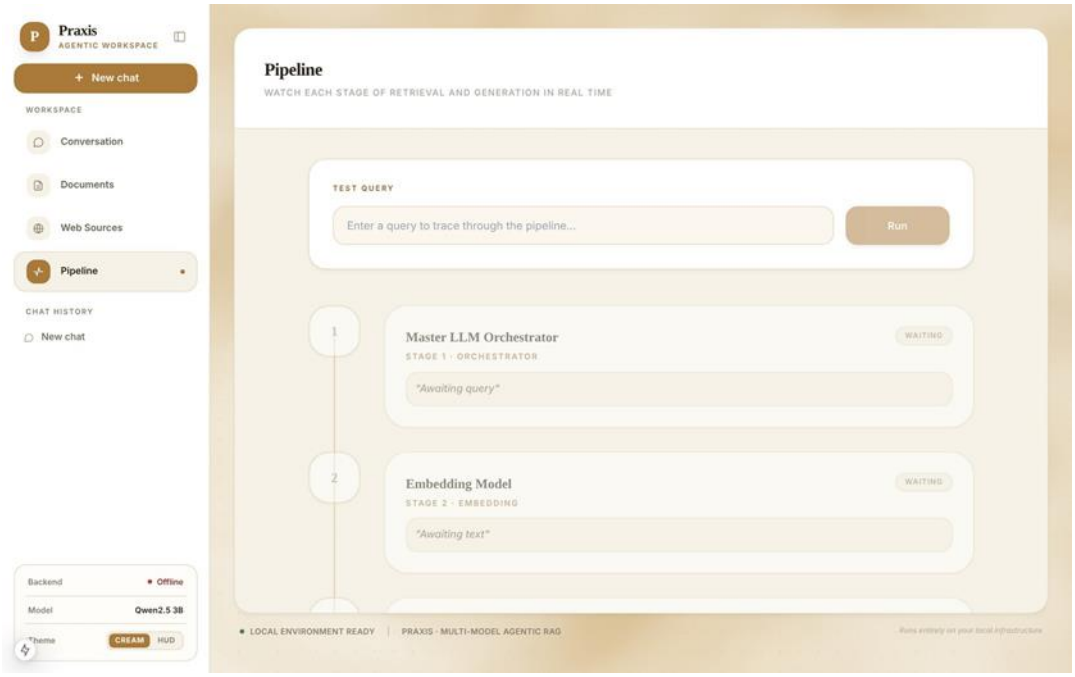


Fig. 7. Pipeline visualizer

H. User Interface and Pipeline Monitoring

The Praxis interface provides a sidebar with a new-chat button, the four workspace views, chat history (the 20 most recent conversations, kept in the browser), backend status, the current model and a theme switch between Cream (light) and HUD (dark). The Conversation view renders markdown answers with inline citations, source links, charts and verification warnings, and its input bar accepts typed or spoken questions with image, document and URL attachments. The Pipeline view traces a test query through seven stages (orchestrator, embedding, vector retrieval, reranking, generation, verification and final response), showing the status and timing of each. Stage events and answer tokens are delivered over Server-Sent Events as soon as they are produced, and request traces are available through /api/traces.

I. Software Requirements

- Operating System: Windows or Linux with Docker and Docker Compose.
- Programming Languages: Python 3.10+ and TypeScript.

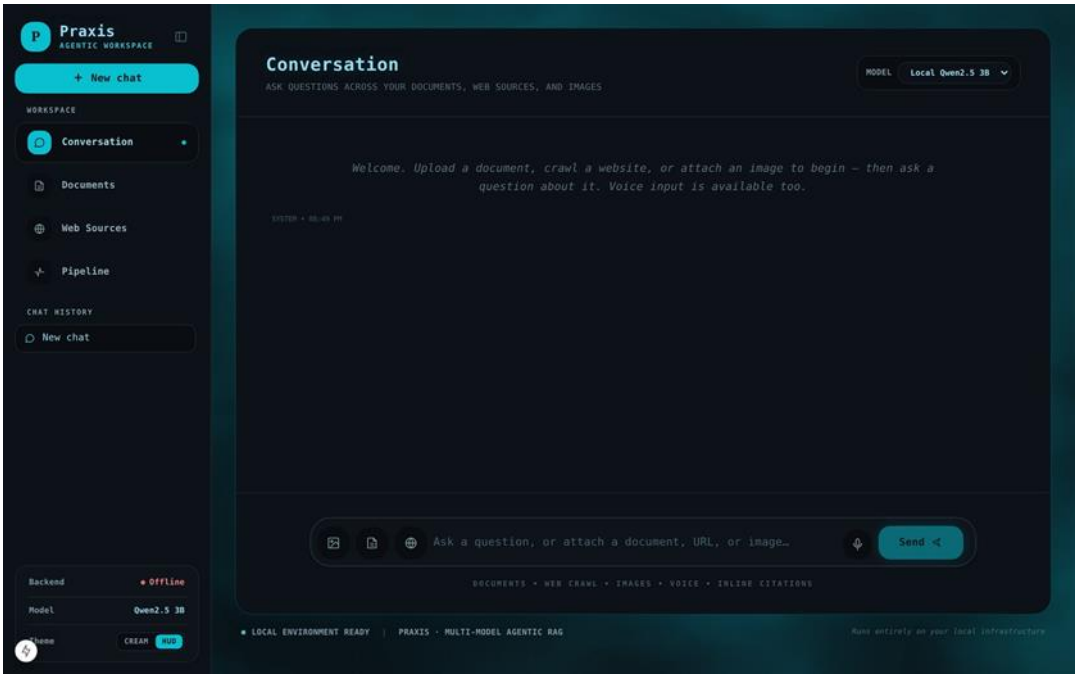


Fig. 8. Conversation view (HUD theme)

- Backend Framework: FastAPI with Uvicorn, Pydantic and sse-starlette.
- Frontend Framework: Next.js 15 with React 18 and Tailwind CSS 3.
- LLM Serving: Ollama (Qwen2.5-3B, Llama 3.2, LLaVA 7B), Gemini API and Claude API.
- Storage: FAISS index on disk, Redis cache and JSON/JSONL files.
- Development Tools: VS Code, Git, pytest, Make and GitHub Actions CI.
- Hardware: Intel Core i5 or equivalent, 16 GB RAM, an NVIDIA GPU with at least 4 GB VRAM (tested on an RTX 2050) and 20 GB of free storage.

J. Libraries and Tools Used

- LangChain for document abstractions, text splitting, FAISS integration and LLM wrappers.
- FAISS and sentence-transformers for dense retrieval and cross-encoder reranking.
- PyPDF, pypdfium2, EasyOCR, BeautifulSoup and Pillow for ingestion, OCR and crawling.
- SearXNG, duckduckgo-search, Wikipedia and Google News for multi-tier web search.
- Pandas, Matplotlib and Seaborn for chart generation.
- Google API client (Gmail) and the Telegram Bot API for approved actions.
- react-markdown with remark-gfm, and Vanta.js/Three.js for the interface.

3. IMPLEMENTATION SCREENSHOTS AND RESULTS

A. Implementation Screenshots

Figs. 4–8 show the implemented Praxis interface. Fig. 4 is the Conversation view in the Cream theme, with the model selector (Local Qwen2.5 3B) and the input bar offering

image, document and URL attachments, voice input and Send. Fig. 5 is the Documents view, where PDF, Markdown or text files are uploaded and ingested into the knowledge base. Fig. 6 is the Web Sources view, which crawls a target URL (single pages and site maps of up to 15 pages). Fig. 7 is the Pipeline view, which traces a test query stage by stage, and Fig. 8 shows the Conversation view in the dark HUD theme.

B. Evaluation Setup

All results were produced by the project's evaluation harness on the development machine, using Qwen2.5-3B through Ollama, all-MiniLM-L6-v2 embeddings, the ms-marco-MiniLM-L-6-v2 cross-encoder and an RTX 2050 GPU with 4 GB of memory. The datasets comprise 64 answerable questions over the project corpus, 24 unanswerable questions, 32 routing cases run three times each, 21 prompt-injection attacks hidden in documents, web pages and PDFs, and 149 traced requests for latency. An answer is counted correct when it contains the gold answer (contains-gold accuracy). For the set Q of questions with gold passage g_q, retrieval is measured by

$$\text{Recall@5} = (1/|Q|)\ \Sigma_q\ 1[g_q \in \text{top-5}(q)] \quad (6)$$

$$\text{MRR} = (1/|Q|)\ \Sigma_q\ 1 / \text{rank}_q(g_q) \quad (7)$$

Improvements were tested for statistical significance with McNemar's test on paired per-question outcomes.

C. Answer Correctness and Retrieval Quality

Fig. 9 shows answer correctness for each pipeline configuration. The model alone answered 3.1% of the questions, while retrieval added about 0.61 in correctness, and the shipped hybrid-plus-rerank stack reached 0.734. The most important single result is the routing fix: before retrieval-first routing the full system answered only 9.4% of the questions correctly because they were sent to web search, and with the knowledge-base probe it reached 67.2%, close to the 73.4% ceiling of the retrieval stack (McNemar $p \leq 0.002$).

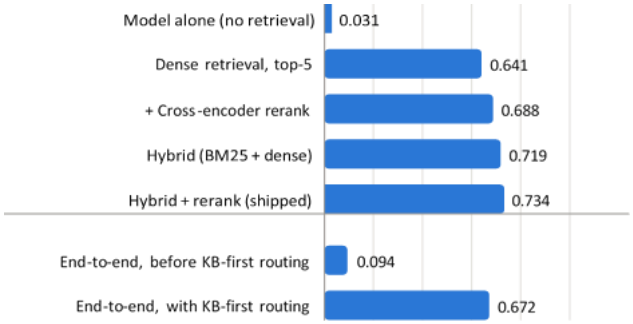


Fig. 9. Answer correctness by pipeline configuration

Fig. 10 compares retrieval configurations. Hybrid retrieval raised recall@5 from 0.781 to 0.938 (p = 0.002), and reranking raised MRR to 0.866 while keeping recall@5 at 0.922. BM25 alone scored highest on this question set because the questions reuse the documents' vocabulary; real users paraphrase, which is why hybrid retrieval remains the default.

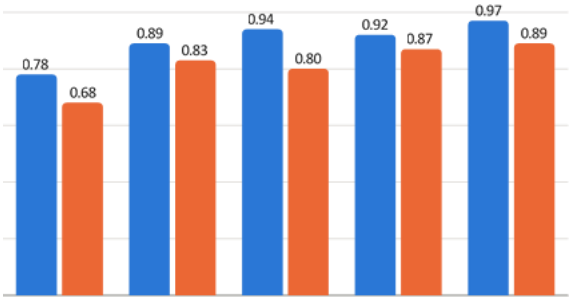


Fig. 10. Retrieval recall@5 and MRR by configuration

D. Verification, Abstention and Security

Fig. 11 shows the rate of correct verifier verdicts before and after the unsupported-specifics check was added. Answers with a fabricated sentence were caught 100% of the time (previously 71%) and answers with a swapped number 100% (previously 33%), while the check itself added only one false alarm in 43 correct answers. With the answer floor, all 24 unanswerable questions were declined (previously 21–22), and only 2 of the 64 answerable questions were declined. In the prompt-injection evaluation the model followed the planted instructions in 11 of 21 attacks, yet end-to-end compromise was zero: no draft was created for an attacker's address, and the allowlist held in all ten attempts that targeted it. Safety enforced in code therefore held even when the model did not.

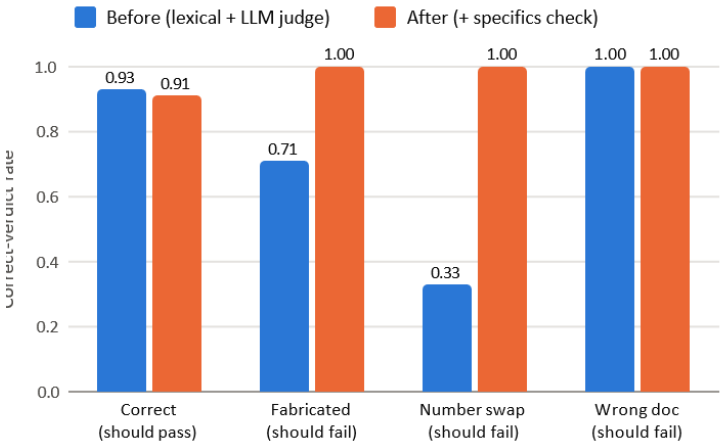


Fig. 11. Verifier accuracy before and after the specifics check

E. Latency and Routing

Over 149 traced requests the median end-to-end latency was 10.4 s and the 90th percentile 21.6 s; knowledge-base answers had a median of 4.3 s and web answers 18.7 s, and a cache hit returned in about 4 ms. Fig. 12 shows that web search and generation dominate the median time per stage. Placing deterministic rules before the classifier raised strict routing accuracy to 0.969 with no unstable cases across three runs, at 0.147 s per case (previously two unstable cases at 1.8 s per case). Table II summarises the results.

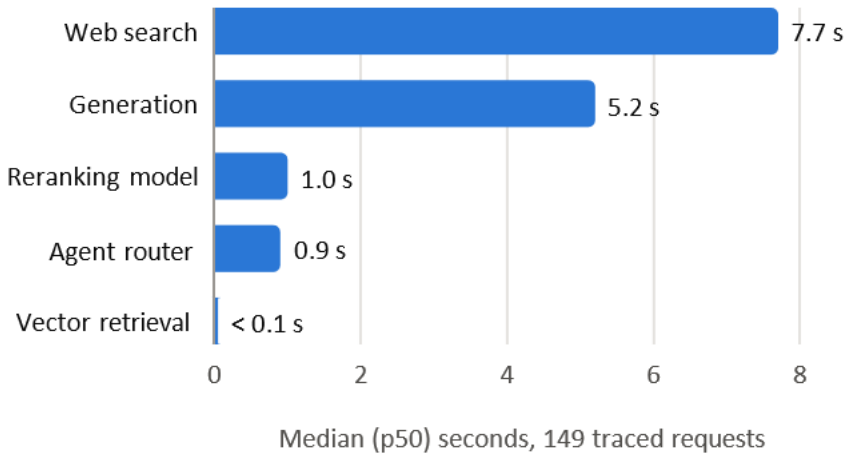


Fig. 12. Median time per pipeline stage

TABLE II. Summary of Evaluation Results

Measure	Result
Routing strict accuracy (32 cases × 3 runs)	0.969, no unstable cases
Retrieval recall@5 / MRR (hybrid + rerank)	0.922 / 0.866
Answer correctness, end-to-end	0.672
Unanswerable questions declined	24 of 24
Fabricated / swapped-number answers caught	100% / 100%
Prompt-injection end-to-end compromise	0 of 21
End-to-end latency p50 / p90 (149 requests)	10.4 s / 21.6 s
Cache-hit latency	≈ 4 ms
Automated tests	349 (338 passed, 11 ML-only tests skipped in CI)

4. FUTURE SCOPE

Several directions can extend Multi-Model RAG. The current deployment is scoped to a single local user, so multi-user authentication with per-user knowledge bases is the first planned extension. The present datasets are modest and project-specific, so a larger, independently written question set would allow the smaller effects observed here to be confirmed, and a stronger judge model could catch

vaguer fabrications that the deterministic checks do not target. The web-search path and OCR ingestion, which the harness does not yet measure, should be added to the evaluation. Retrieval can be extended to images and charts inside documents, domain-specific fine-tuned models can be evaluated against the general-purpose ones used now, and the system can be deployed to the cloud using the AWS architecture already designed for it.

5. CONCLUSION

This paper presented Multi-Model RAG, an agentic retrieval-augmented generation system that decides how each question should be answered, whether from the user's documents, the live web, an image, a chart or simple conversation, and then streams a grounded answer with citations and a live view of every pipeline stage. The system integrates three-level routing, hybrid FAISS and BM25 retrieval with reciprocal rank fusion, cross-encoder reranking with an answer floor, OCR-aware ingestion, multi-tier web search, vision analysis, local and API language models, fast answer verification and approval-

gated actions in a local-first application that runs on a 4 GB laptop GPU. On the project's evaluation harness, hybrid retrieval raised recall@5 from 0.781 to 0.938, retrieval-first routing raised end-to-end correctness from

0.094 to 0.672, every unanswerable question was declined, and every fabricated sentence and swapped number in the verifier test set was flagged. Although the model obeyed planted instructions in about half of the prompt-injection attacks, none produced a dispatchable message. Multi-Model RAG thus provides a practical foundation for trustworthy assistants that show where their answers come from, admit when they do not know, and never act on the user's behalf without permission.

6. REFERENCES

[1] P. Lewis, E. Perez, A. Piktus *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459–9474.

[2] V. Karpukhin, B. Oğuz, S. Min et al., "Dense passage retrieval for open-domain question answering," in Proc. Conf. Empirical Methods in Natural Language Processing (EMNLP), 2020, pp. 6769–6781, doi: 10.18653/v1/2020.emnlp-main.550.

[3] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: BM25 and beyond," Foundations and Trends in Information Retrieval, vol. 3, no. 4, pp. 333–389, 2009, doi: 10.1561/1500000019.

[4] G. V. Cormack, C. L. A. Clarke, and S. Büttcher, "Reciprocal rank fusion outperforms Condorcet and individual rank learning methods," in Proc. 32nd Int. ACM SIGIR Conf. Research and Development in Information Retrieval, 2009, pp. 758–759, doi: 10.1145/1571941.1572114.

[5] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in Proc. EMNLP-IJCNLP, 2019,

[6] pp. 3982–3992, doi: 10.18653/v1/D19-1410.

[7] R. Nogueira and K. Cho, "Passage re-ranking with BERT," arXiv:1901.04085, 2019.

[8] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," IEEE Transactions on Big Data, vol. 7, no. 3, pp. 535–547, 2021, doi: 10.1109/TBDATA.2019.2921572.

[9] S. Yao, J. Zhao, D. Yu et al., "ReAct: Synergizing reasoning and acting in language models," in Proc. Int. Conf. Learning Representations (ICLR), 2023.

[10] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi, "Self-RAG: Learning to retrieve, generate, and critique through self-reflection," in Proc. Int. Conf. Learning Representations (ICLR), 2024.

[11] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert, "RAGAS: Automated evaluation of retrieval augmented generation," in Proc. 18th Conf. European Chapter of the Association for Computational Linguistics (EACL): System Demonstrations, 2024, pp. 150–158, doi: 10.18653/v1/2024.eacl-demo.16.

[12] K. Greshake, S. Abdelnabi, S. Mishra, C. Endres, T. Holz, and M. Fritz, "Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection," in Proc. 16th ACM Workshop on Artificial Intelligence and Security (AISec), 2023, pp. 79–90, doi: 10.1145/3605764.3623985.

[13] Qwen Team, "Qwen2.5 technical report," arXiv:2412.15115, 2024.

[14] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," in Advances in Neural Information Processing Systems (NeurIPS), vol. 36, 2023.