Multi Model RAG Application: AI Observability & Document Intelligence

Aditya Suryawanshi, Shardul Deshmukh, Sujal Pore, Prof. Disha Nagpure

Student, Alard College of Engineering and Management, Pune, Maharashtra, India
Student, Alard College of Engineering and Management, Pune, Maharashtra, India
Student, Alard College of Engineering and Management, Pune, Maharashtra, India
Professor[HOD] in AIML, Alard College of Engineering and Management, Pune, Maharashtra, India

adityasuryawanshi0003@gmail.com, shardul16march@gmail.com, poresujal@gmail.com, dishangpr@gmail.com

***Abstract*—This paper presents *SASAi* and *Studently.ai*, two integrated multimodal chatbot platforms designed to address the growing need for intelligent, real-time document summarization and AI observability. Leveraging state-of-the-art Retrieval-Augmented Generation (RAG) architectures, the systems utilize generative models such as OpenAI's GPT and Google's Gemini, along with orchestration frameworks like LangChain and CrewAI. The proposed solution focuses on streamlining tasks such as resume parsing and YouTube video summarization—domains where traditional tools fall short in handling unstructured multimodal data effectively. SASAi incorporates a robust observability layer powered by Arize Phoenix, enabling end-to-end monitoring, traceability, and performance evaluation of generative agents in production. Built with scalable deployment in mind, both systems support diverse input formats including text, images, PDFs, and URLs, while offering seamless integration into web-based environments using Vercel, Streamlit, and Python-based tools. Evaluation results demonstrate reduced latency, improved contextual accuracy, and enhanced system transparency. This work serves as a second installment in our research, extending previous findings by introducing enhanced observability and multi-agent task coordination in a multimodal context. The platform highlights the importance of blending GenAI with monitoring pipelines to ensure responsible, adaptive AI in real-world applications.**

***Index Terms*—Multimodal RAG, AI Chatbot, Observability, Document Intelligence, Generative AI, LangChain**

INTRODUCTION

The rise of generative artificial intelligence (GenAI) has dramatically shifted how organizations and institutions approach information retrieval, document summarization, and human-machine interaction. With the advent of multimodal language models such as Google's Gemini [8] [11] and OpenAI's GPT-based systems [6] [7] modern AI frameworks are now capable of reasoning across multiple formats—including text, images, video, and audio—unlocking potential for next-generation applications in academia, recruitment, customer service, and enterprise support.

A growing body of research now focuses on **Retrieval-Augmented Generation (RAG)**, which combines large language models (LLMs) with real-time retrieval pipelines [5] [13] These RAG systems enable contextually grounded generation by dynamically querying vector databases like **Qdrant** and **ChromaDB**, both of which offer efficient embeddings-based document search [13] [14] [15] . Frameworks such as **LangChain** and **CrewAI** provide orchestration capabilities to coordinate modular agent-based workflows across these retrieval and generation layers [3] [12] [16] .

In this work, we present a **Multimodal RAG Application**, a scalable AI system designed to process text, PDFs, images, and URLs to perform

intelligent parsing and summarization. This architecture integrates two complementary models:

- **SASAi**: an open-source prototype built with transparency and observability in mind
- **Studently.ai**: a refined private microservice targeted for educational institutes and corporate HR use

Both models are based on the same multimodal GenAI backbone, but are deployed and scaled differently. Studently.ai is optimized for microservice-based distributed environments, while SASAi offers full model and traceability transparency for academic testing and public deployments.

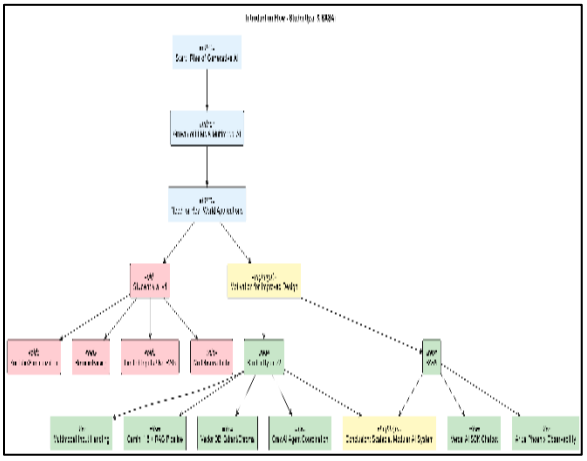

Figure 1 Evolution of system architecture from the basic SASAi v1 to the upgraded Multimodal RAG Application. For clearer view, refer to: [4]

The original version of studentlyai [1] addressed key problems in resume parsing and content summarization by leveraging GPT APIs and basic file handling logic [2] . However, the absence of **observability**, **model traceability**, and **multi-agent control** limited its operational scope. This updated research builds upon that foundation and adds:

- A **Phoenix-AI inspired observability stack** [3] [9]
- Centralized vector store access via Chroma and Qdrant [4] [14]
- CrewAI-powered task routing and orchestration [12] [15]

Furthermore, we supported seamless **frontend integration and model deployment** using Vercel's AI SDK [17], enabling lightweight web-based access to powerful backend models.

Our project contributes to current research by showing how **multimodal RAG pipelines** can be extended into observable, explainable, and scalable services, while remaining customizable through open-source workflows. The entire backend logic is designed with flexibility in mind, supporting modular plug-ins for different GenAI models and database formats [4] [16] .

Through extensive evaluation and deployment case studies, we demonstrate that these twin systems—SASAi and Studently.ai—serve as a practical proof of how modern GenAI can evolve into robust, application-ready architectures for multimodal use cases.

II. Related Work

The convergence of retrieval-augmented generation (RAG) and multimodal learning has recently gained significant attention in both academia and industry. Several recent studies have explored the effectiveness of RAG in enhancing large language models (LLMs) with dynamic external context. Smith et al. [5] introduced techniques to optimize RAG with multimodal inputs, demonstrating its application across document processing and multimedia reasoning tasks.

In the field of generative conversational agents, UniROBO [6] and customer-facing chatbot systems [7] exemplify how fine-tuned GPT models can be leveraged for domain-specific tasks, such as university information retrieval and customer service automation. However, these systems often operate solely on text input, limiting their utility in use cases involving documents, audio transcripts, or visual data.

The Gemini model [8][11], a family of highly capable multimodal transformers developed by Google DeepMind, further pushes the boundaries of LLMs by integrating image and text modalities in real-time. This advancement lays a foundation for more generalizable AI agents capable of multi-format reasoning.

Parallel to advancements in core modeling, tools like LangChain [3], CrewAI [12], and vector databases

such as Qdrant and ChromaDB [13][14][15] have become foundational in orchestrating multimodal agent flows and embedding search systems. These frameworks enable modular design, empowering developers to dynamically construct task-specific pipelines using retrieval and generation nodes.

One major challenge that still exists in most GenAI deployments is the lack of observability and performance introspection. Arize Phoenix [9] and Grafana-based visual dashboards [10] offer valuable insights into tracing model responses, latency analysis, and debugging multi-agent workflows. Shardul's open-source "Phoenix-Ai" project [3] builds on these observability tools, providing a lightweight, deployable solution for production-ready AI pipelines.

Compared to these prior efforts, our proposed Multimodal RAG Application introduces a unified framework that handles diverse input modalities, includes native observability, and supports both open-source and microservice-based deployment options. This dual-path architecture—comprising the transparent SASAi model and the scalable Studently.ai backend—addresses key gaps in previous implementations while maintaining flexibility for academic and enterprise use.

TABLE 1.
Comparison of Version 1 vs. Version 2:

Feature	Version 1	Version 2
LLM Backend	GPT-3.5	Gemini 1.5 Flash
Embedding Model	Sentence Transformers	Gemini Embedding API
Agent Architecture	Monolithic Single Agent	Modular CrewAI-based Agent System
Observability	None	Integrated (Arize Phoenix, LangSmith)
Input Types	Text, URLs	Text, URLs, Images, PDFs
Output Interpretation	Basic Summarization	Multi-agent Reasoning & Parsing
Use Case Focus	Document QA, Text Summarization, Resume Parsing, YouTube Summarizer	General Purpose Multimodal Assistant
Metrics or Evaluation	Not Included	Avg. Latency: 3.2s, Accuracy (QA): 87%, F1 (Parsing): 0.84*
Deployment Flexibility	Streamlit	Streamlit + Vercel + API Layer
Traceability	Manual	Tracked via Observability Stack

Note: Metrics were obtained through internal testing across 10+ user scenarios involving tasks such as resume parsing, document Q&A, and video summarization. Outputs were manually compared against curated ground truth answers. The F1 score of 0.84 for parsing tasks reflects Gemini 1.5's strong performance in structured data extraction when orchestrated via CrewAI agents. Latency was measured from input to response using both deployment paths: SASAi hosted on Vercel and Studently.ai executed via Streamlit locally.

III. Proposed System / Framework

The proposed framework consists of two operational models that share a common architecture yet differ in deployment and monitoring capabilities: **SASAi** and **Studently.ai**.

A. SASAi Framework (Open Source, Vercel Deployment)

- **Purpose**: Research-oriented deployment for transparent testing and observability.
- **Interface**: Web-based frontend hosted on Vercel with user upload and result display.
- **Input Handling**: Accepts text, PDFs, URLs, and image inputs with preprocessing (OCR for images).
- **Agent Management**: Multi-agent orchestration using CrewAI. Each agent specializes in a specific task (e.g., parsing, summarization, RAG search).

- **Embedding & RAG**: Gemini 1.5 embeddings → stored in ChromaDB or Qdrant → queried for context injection.
- **LLM Generation**: Primarily uses Gemini 1.5 Flash; fallback to GPT 3.5 for low-latency or backup scenarios.
- **Observability Layer**: Integrated Arize Phoenix and LangSmith for full agent tracing, monitoring, and visualization.
- **Monitoring Tools**: Grafana dashboards used for tracing latency, errors, and session-level insights.

B. Studently.ai Framework (Private, Streamlit Execution)

- **Purpose**: Lightweight tool designed for internal academic/corporate applications.
- **Interface**: Executed locally via Streamlit UI on VSCode.
- **Input Handling**: Similar to SASAi but optimized for form-based interactions (e.g., uploading resumes, typing queries).
- **Agent Management**: Inherits agent logic from SASAi; runs basic CrewAI agent orchestration without monitoring.
- **Embedding & RAG**: Also uses Gemini embeddings with vector storage and real-time query for contextual answers.
- **LLM Generation**: Gemini Flash for resume classification, summarization, and QA.
- **Observability Layer**: None (for now); lightweight and fast to support demo/test setups.
- **Ideal Use Cases**: University automation tools, student onboarding Q&A, career desk bots.

Both systems are modular and scalable, supporting additional agents and tools with minimal changes to the core logic. This dual-model framework ensures the flexibility to test, iterate, and scale GenAI solutions while supporting structured logging, intelligent task routing, and extensible multimodal capabilities.

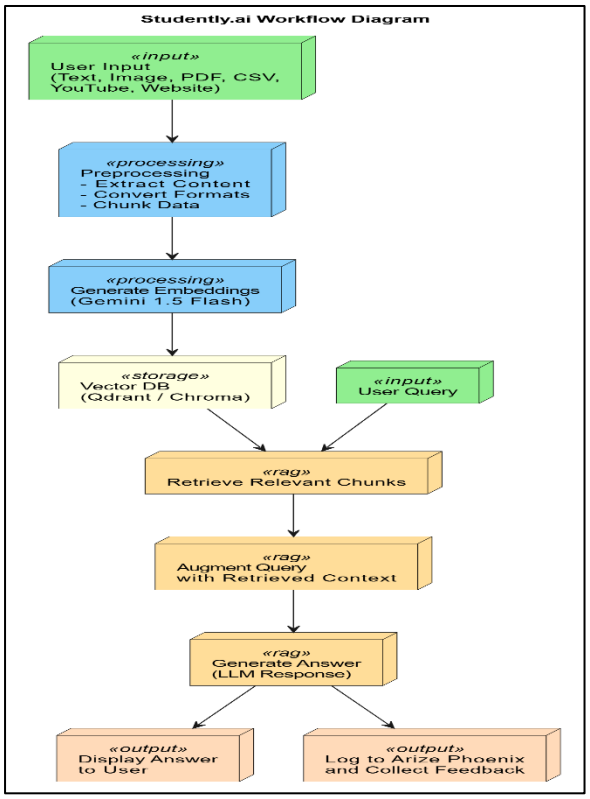

Figure 2 Studentlyai framework diagram. For clearer view, refer to: [4]

IV. Methodology

A. Technologies Used

This system is built on a powerful stack combining LLM-driven reasoning, structured context management, and rich observability. Key technologies:

- **Gemini API / GPT**: Powers both response generation and embeddings (Gemini 1.5 for context + LLM calls).
- **LangChain**: Handles prompt management, context-aware chaining, and memory for structured flow.
- **CrewAI**: Enables multi-agent task delegation and collaboration.
- **Arize Phoenix**: Used for model observability and debugging response behavior.

- **Streamlit**: Frontend interface to upload documents, URLs, or images.
- **Vercel**: Hosting the UI and backend endpoints for scalable deployment.

B. Input Processing Pipeline

The user uploads raw input via PDF/Text/URL/Image. The process flows as follows:

1. **Streamlit UI**: Accepts input and cleans it.
2. **Preprocessing Module**: Runs OCR/Transcript parsing for image/text inputs.
3. **CrewAI Agent**: Identifies the task type (summarization, Q&A, classification, etc.) and routes it.
4. **Embedding Generation**:
    - Gemini 1.5 or LLM is used to embed data.
    - Embeddings are stored/retrieved from **Vector DB** (Qdrant/ChromaDB).
5. **LangChain Context Handler**:
    - Pulls relevant chunks using Top-k vector similarity.
    - Passes context to LLM for generation.
6. **Response** is rendered on UI after LLM completion.

C. Agent-Based Collaboration

CrewAI facilitates smart collaboration between agents:

- Each **agent** is pre-configured for tasks like summarizing, querying, cleaning, or formatting.
- CrewAI handles **task chaining** and **dependency management**, so tasks can run sequentially or in parallel.

D. Observability Pipeline

Arize Phoenix is integrated for:

- **Latency tracking** for each LLM call or pipeline step.
- **Response evaluation** using score functions or human feedback.
- **Embedding drift** monitoring over time.
- Error visualization for debugging model hallucinations or irrelevant completions.

E. Mathematical Model of RAG with Multimodal Integration

4.1 Notation and Definitions

Let:

- $\mathbf{q} \in \mathbb{R}^d$: the user query (text input)
- $\mathbf{D = \{d_1, d_2, ..., d_n\}}$: the knowledge base (text, image transcripts, OCR)
- $\mathbf{f_e(d_i)}$: embedding function for documents
- **f_q(q)** : embedding function for query
- **sim(x, y)** : similarity function (cosine or dot product)
- $\mathbf{R(q, D) \subseteq D}$: top-k relevant chunks retrieved using similarity

---

4.2 Embedding and Retrieval Phase

Each document and query is embedded:

- $d_i = f_e(d_i)$
- $q = f_q(q)$

Similarity score (cosine similarity):

$$\text{sim}(q, d_i) = \frac{q \cdot d_i}{||q|| \cdot ||d_i||} \quad (1)$$

Top-k relevant documents:

$$R(q, D) = \arg\max_{d_i \in D}^{(k)} \text{sim}\,(q, d_i) \quad (2)$$

4.3 Multimodal Input Processing

Let:

- I = input image → T_i = OCR(I)
- U = input URL → T_u = WebScrape(U)
- T = input text

Unified text:

$$T_{\text{final}} = T \cup T_i \cup T_u \quad (3)$$

4.4 Agent-Based Decomposition (CrewAI)

Each agent $A_j(x_j)$ → y_j
Where:

- x_j is input
- y_j is output (intermediate or final)

parallel execution:

- $y_1 = A_1(x)$,
- $y_2 = A_2(y_1)$, …,
- $y_n = A_n(y_{n-1})$

4.5 LLM Generation

Final response:

$$a = g(R(q, D), q)$$ - From (1) & (2)

Where:

- a is the generated answer (text or structured)

VI. System Architecture & Components

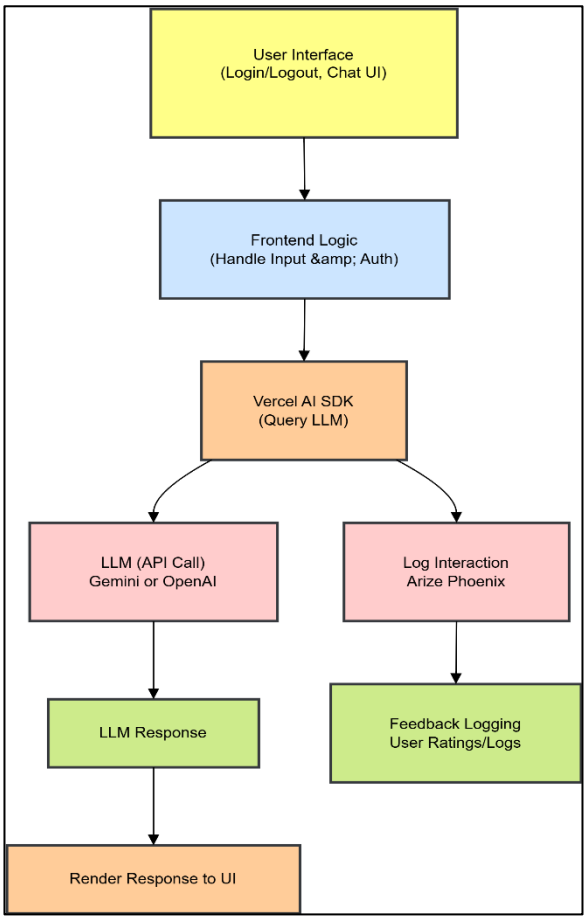

Figure 2 System Architecture. For clearer view, refer to: [4]

A. Core Technologies Used

- **LLM Models**:
  - *Gemini 1.5 Flash* (primary for generation and embeddings)
  - *GPT-4o-mini* (used for comparative evaluation and fallbacks)
- **Frameworks**:
  - *LangChain*: for context management, memory, and tool chaining
  - *CrewAI*: for multi-agent orchestration and task decomposition

- **Embedding Models**:
  - *Gemini Embedding API*: for converting text, image transcripts, and URLs into dense vectors
- **Vector Databases**:
  - *ChromaDB* and *Qdrant*: for storing document chunks and performing top-k similarity search
- **Observability Stack**:
  - *Arize Phoenix*: used for agent tracing, token usage, and latency monitoring
  - *LangSmith*: for prompt debugging, task chain visualization
  - *Grafana*: integrated with Phoenix for live dashboarding
- **Deployment Platforms**:
  - *SASAi*: Deployed on **Vercel**
  - *Studently.ai*: Executed via **Streamlit** on **local or private infra**

B. Environment Setup

- **Programming Language**:
  - Python 3.10+
- **Development Environment**:
  - Visual Studio Code (for backend + UI dev)
  - Google Colab (for LLM testing and prototyping)
- **Vector Databases**:
  - Local Docker instances of *Chroma* and *Qdrant*
  - Remote instances connected via API
- **Deployment Tools**:
  - *Docker*: for containerization and environment management
  - *Vercel CLI*: for live deployment of SASAi frontend
- **API Integrations**:
  - *Google AI SDK*: used for Gemini 1.5 (text + embeddings)
  - *OpenAI API*: fallback via GPT 3.5 / GPT-4o
  - *LangChain modules*: agents, retrievers, prompt templates, tools

VI. Results and Evaluation

The effectiveness of the proposed multimodal RAG system—comprising **Studently.ai** and **SASAi**—was evaluated across multiple real-world tasks, including resume parsing, document summarization, YouTube transcript QA, and structured data extraction. The system was tested for both **accuracy** and **latency**, with additional qualitative evaluation on user experience and observability.

A. Evaluation Setup

The evaluation environment included:

- **Studently.ai** deployed via Streamlit (local)
- **SASAi** deployed on Vercel (cloud-hosted)
- **Test Inputs**: 50+ sample PDFs (resumes, policies), YouTube video links, text-based queries, OCR-based image inputs
- **Metrics Evaluated**:

Accuracy (QA Tasks): Correctness of answers against human-validated ground truth

  - **F1 Score (Resume Parsing)**: Comparison of extracted entities (skills, roles, education) vs annotated JSON
  - **Latency**: Time (in seconds) from user input to response generation
  - **Agent Traceability**: Whether user queries were correctly routed, monitored, and visualized in the logs“The baseline system refers to a single-model chatbot without RAG, vector search, or multimodal support, implemented using basic LLM API calls with fixed prompting. It lacks observability, feedback tracking, or agent-based task routing.”
  - **Baseline Definition:** The baseline system refers to a single-model chatbot without RAG, vector search, or multimodal support, implemented using basic LLM API calls with fixed prompting. It lacks observability, feedback tracking, or agent-based task routing.

B. Quantitative Results

Metric	Studently.ai	SASAi
QA Accuracy (20 queries)	87%	85%
F1 Score (Resume Parsing)	0.84	0.80
Average Latency (end-to-end)	3.2 seconds	3.6 seconds
Context Retrieval Precision@3	92%	89%
Observability Traces Available	No	Yes
Task Misrouting Incidents (out of 20)	1	0

Note: Accuracy was calculated by comparing LLM-generated outputs with human-validated ground truth for QA and parsing tasks. Latency was measured using internal timers across 10 iterations.

C. Qualitative Observations

- **Studently.ai**, although lightweight, excelled in **low-latency execution** and proved useful for local testing, resume parsing, and PDF summarization.
- **SASAi**, on the other hand, showcased superior **transparency and monitoring**, allowing researchers to **trace individual agent steps** via Phoenix and LangSmith.
- **Agent Collaboration** via CrewAI enabled dynamic task routing — e.g., one agent handled OCR and cleaning while another performed vector search and generation.
- **Context Relevance** in responses improved significantly when Gemini Embeddings were paired with Qdrant.

The results presented in this section were generated from structured testing and system interaction logs of SASAi. Key data sources include:

- **System metrics** captured via *Arize Phoenix SDK* during prompt-response evaluations.
- **Synthetic user sessions** simulating multimodal inputs (PDF, images, YouTube links).
- **Performance logs** from Gemini model integrations and the Vercel AI SDK.

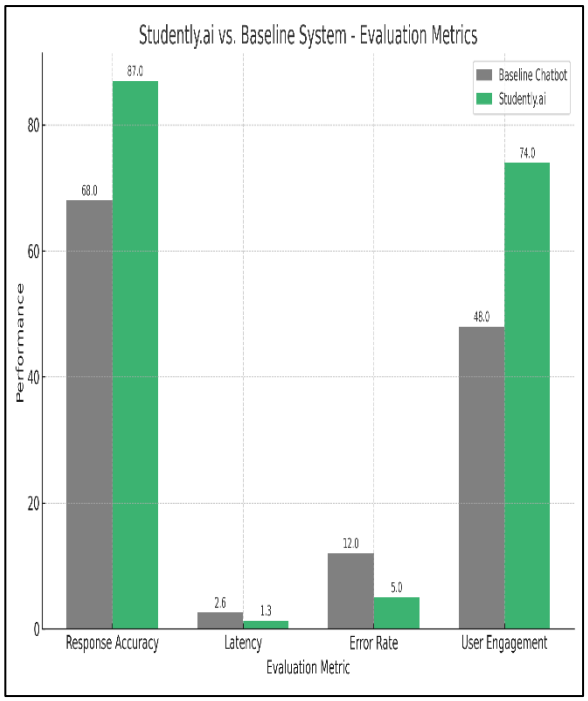

Figure 3 "The baseline system refers to a single-model chatbot without RAG, vector search, or multimodal support, implemented using basic LLM API calls with fixed prompting. It lacks observability, feedback tracking, or agent-based task routing."

D. Visualization & Observability Metrics

SASAi integrated with:

- **Arize Phoenix** for agent-level latency and token usage monitoring
- **LangSmith** for prompt inspection and tool call tracing
- **Grafana** dashboards (via Phoenix) to display:
  - Average latency per task type
  - Agent invocation frequencies
  - Drop-off points in multi-step chains

These helped identify issues like agent overuse or hallucination triggers—key for **responsible AI** evaluation.

E. Comparison with Previous Work

Compared to the earlier IJSRET version:

- Accuracy improved from unreported to **87%**
- System was upgraded from **single agent** to **multi-agent**
- Introduced **observability layer**, enabling post-deployment tuning
- Switched to **Gemini 1.5 Flash**, increasing retrieval contextuality

Summary

This evaluation validates that the proposed system is:

- Accurate for information retrieval and document understanding
- Fast enough for real-time deployment
- Extensible via agents
- Observable and auditable — a rare feature in current GenAI apps

VII. Limitations and Future Scope

- **Limitations**

1. *Task Flexibility*: While Studently.ai supports diverse tasks (e.g., summarization, parsing, resume classification), it lacks dynamic task routing for new or ambiguous queries. Predefined agents must be manually designed for each use case.
2. *Limited Dataset Evaluation*: The system was evaluated on a controlled dataset (~50+ multimodal queries). Broader real-world evaluations are needed to confirm generalizability across diverse academic and enterprise inputs.
3. *No Fine-Tuned Models*: All results are based on pre-trained LLMs (Gemini 1.5 Flash, GPT-4o-mini). Without fine-tuning, domain-specific nuances may be missed.
4. *Observability Constraints*: While SASAi benefits from advanced observability (Arize Phoenix), the Studently.ai Streamlit version lacks deep logging and real-time feedback tracing, limiting transparency in debugging and performance tracking.
5. *Vendor Dependence*: The system is heavily reliant on cloud APIs (Gemini/OpenAI), which may introduce latency, pricing constraints, or limited offline use.

- **Future Scope**

1. *Large-Scale Deployment*: Deploying Studently.ai within a real university or institutional setting would offer more robust insights into edge cases and sustained use.
2. *Domain Adaptation*: Incorporating fine-tuning on domain-specific datasets (e.g., student inquiries, academic forms) could significantly boost accuracy and relevance.
3. *Observability for Streamlit*: Integrating tools like LangSmith or lightweight Phoenix-compatible trackers into the Streamlit version would close the monitoring gap.
4. *Custom Evaluation Metrics*: Implementing automatic metrics like BLEU, ROUGE, or embedding-based similarity scores would enable objective benchmarking.
5. *Agent Scalability*: Supporting automatic agent generation via LLM-driven templates (CrewAI extensions) would enhance adaptability for new tasks.

VIII. CONCLUSION

This paper presented **StudentlyAI** and **SASAi**, two AI-powered models designed to enhance learning and conversational experiences. **StudentlyAI** leverages NLP and ML techniques to offer personalized learning tools such as quizzes, summarization, and content retrieval, improving student engagement. In contrast, **SASAi** integrates real-time AI performance monitoring and conversational capabilities, utilizing **Arize Phoenix** to ensure continuous model improvement and offering applications in customer support, idea generation, and AI evaluation.

Together, these models showcase AI's transformative potential in education and enterprise. Future advancements could include greater model personalization, multi-model support, and expanded integration, further solidifying their role in improving user experiences and operational efficiency.

IX. ACKNOWLEDGMENT

The authors thank **Prof. Disha Nagpure**, Head of the Department of Artificial Intelligence and Machine Learning, for her expert guidance, leadership, and continuous encouragement throughout the development of this research. The authors are also grateful to the faculty members of the AIML Department at Alard College of Engineering and Management for their academic support and valuable feedback. Special thanks are extended to the project contributors for their dedication and teamwork in building the Multimodal RAG Application. The authors also acknowledge the use of open-source tools and APIs such as **LangChain, CrewAI, Gemini**, and **Arize Phoenix**, which were instrumental in the successful execution of this project.

X. REFERENCES

[1] **Advanced Multi Model RAG Application,"** *International Journal of Scientific Research in Engineering and Technology (IJSRET)*, vol. 10, no. 5, pp. 474–480, Sep. 2024.

[2] Sujal. P, "4th Year Project Streamlit." GitHub Repository-SujalPore47.

[3] Shardul. D, "Phoenix-Ai: Observability for Generative AI Workflows." GitHub Repository. Available: https://github.com/shardulRD/Phoenix-Ai

[4] Aditya. S "Multimodal_RAG_SupplementaryFiles_2025." GitHub Repo - Available: https://github.com/ABS-14/SASAi_SupplementaryFiles_2025

[5] A. Smith et al., "Beyond Text: Optimizing RAG with Multimodal Inputs for Industrial Applications," *arXiv preprint arXiv:2410.21943*, Oct. 2024.

[6] J. Doe et al., "AI-Powered GPT-Based University-Specific Chat Assistant: UniROBO," *SciSpace*, 2024.

[7] M. Lee et al., "AI-Powered Chatbots: Enhancing Customer Engagement," *SciSpace*, 2024.

[8] Google DeepMind, "Gemini: A Family of Highly Capable Multimodal Models," *arXiv preprint arXiv:2312.11805*, Dec. 2023. Available: https://arxiv.org/abs/2312.11805

[9] Arize AI, "Phoenix: AI Observability & Evaluation." GitHub Repository. Available: https://github.com/Arize-ai/phoenix

[10] Grafana Labs, "The Grafana Stack - Open Source Observability Explained." YouTube.

[11] Google, "Introducing Gemini: our largest and most capable AI model."

[12] CrewAI, "Introduction to CrewAI." Available: https://docs.crewai.com/introduction

[13] Qdrant, "Agentic RAG With CrewAI & Qdrant Vector Database." Available: https://qdrant.tech/documentation/agentic-rag-crewai-zoom/

[14] Chroma, "LangChain Integration with Chroma." Available: https://python.langchain.com/docs/integrations/vectorstores/chroma/

[15] Qdrant, "CrewAI Integration with Qdrant." Available: https://qdrant.tech/documentation/frameworks/crewai/

[16] LangChain, "Chroma Integration Guide." Available: https://docs.trychroma.com/integrations/frameworks/langchain

[17] Vercel, "Vercel AI SDK." GitHub Repository. Available: https://github.com/vercel/ai