Final report

CS-671 Deep Learning And Applications
Hackathon

MATBOT AI Assistant - AI Assistant and Troubleshooter for MATLAB Related Queries

Submitted By:
Group - 21

Contents

1 Approach Outline

1.1 Problem Understanding

In the evolving field of Generative AI, Retrieval-Augmented Generation (RAG) systems must address complex queries autonomously and efficiently. The Agentic RAG system seeks to enhance RAG capabilities by introducing greater autonomy in retrieval, data synthesis, and decision-making. Key challenges include:

- **Autonomous Decision-Making**
  The agent must dynamically determine optimal retrieval and generation strategies, analyzing query complexity and resource use to maximize cost-effectiveness without compromising accuracy.
- **Contextual Relevance and Accuracy**
  Maintaining high contextual relevance and accuracy requires the agent to filter and rank information from multiple sources, ensuring aligned, precise responses, even with unstructured data.
- **Error Resilience and Adaptability**
  The agent should be resilient to external API errors or retrieval issues, seamlessly switching to alternative sources to maintain performance and reliability.
- **Resource Efficiency and Scalability**
  Efficient resource use is essential, particularly for handling large data volumes and high-frequency queries. Load balancing, caching, and optimized pathways support real-time scalability.
- **Query Comprehension and Intent Recognition**
  The system must accurately interpret user queries, understanding intent, domain context, and any implicit constraints to guide correct retrieval and response generation.
- **Knowledge Base Enhancement and Data Formulation**
  Maintaining an up-to-date and structured knowledge base is critical. The system should continuously ingest and vectorize reliable content, enriching metadata to improve retrieval performance.
- **Feedback Mechanism for Continuous Improvement**
  A structured feedback loop allows users to evaluate solutions. This input can be logged to enhance agent performance, prioritize content updates, and refine response quality over time.
- **Evaluation Scheme and Testing**
  The application should be validated using diverse troubleshooting scenarios. Evaluation should consider accuracy, user satisfaction, and adaptability to evolving documentation and problem types.

This project aims to create an agentic RAG-based troubleshooting assistant focused on MATLAB-related issues. It will integrate: dynamic retrieval strategies, a vectorized knowledge base from MATLAB documentation, error handling with source fallback, interactive feedback collection, and clear step-by-step guidance for users through a Gradio or Streamlit interface.

1.2 Solution Overview

Our solution for troubleshooting utilizes an AI-driven, multi-step approach, structured as follows:

- **Query Enhancement**: Given a user query, the first step is to rewrite the query to generate more comprehensive and compact versions. We leverage the *Query Rewrite* method to enhance the clarity of the query and then apply the *Hyde* method to make the query more detailed and broad, ensuring a better understanding by the system.
- **Vector Store**: We convert the input data into a structured JSON format, which is then chunked into coherent data segments rather than relying on random token-sized chunks. This structured approach is critical in enhancing the efficiency of information retrieval. To reduce search complexity and improve retrieval speed, we utilize the *DiskANN* method, which clusters these chunks effectively.

- **Image Searching**: The system also facilitates the search for relevant images. We utilize CLIP (Contrastive Language-Image Pretraining) embeddings to construct an image database, which is subsequently used to retrieve images based on similarity searches.
- **Feedback Loop**: After providing an initial AI-generated response, we integrate a feedback mechanism where the user can evaluate the answer. If the user is not satisfied, the response is regenerated, taking into account metrics such as context relevance, accuracy, and other parameters. This process incorporates human-in-the-loop and self-regenerative methods (Self RAG) to continuously improve the responses based on user feedback.
- **Evaluation**: We employ the *RAGAS Evaluation Criteria* to judge the AI's response on key parameters such as context relevance, answer relevance, and faithfulness. This ensures that the AI-generated solutions meet high standards of precision and relevance.
- **Self Memory**: A dynamic vector database is maintained, storing each query-response-context pair. This enables a self-memory mechanism that improves future responses by reusing relevant historical context, making the system adaptive and continually improving.
- **Interactive Streamlit Frontend**: The system features an interactive frontend using Streamlit, allowing users to input their queries, view generated troubleshooting steps, and provide feedback. This UI offers a smooth experience, enhancing user engagement.
- **VS Code Extension**: A VS Code extension is developed to integrate the troubleshooting solution directly within the development environment. This extension offers seamless query submission, real-time results, and the ability to interact with the system without leaving the coding workspace.

1.2.1 Core Concepts

- **Query Enhancement**: Refine user input through query rewriting and broaden the scope of the query to improve AI understanding and accuracy.
- **Vector Store and Chunking**: Transforming input data into structured chunks, clustered using DiskANN, to improve search efficiency and retrieval accuracy.
- **Feedback Loop and Self-RAG**: Incorporating user feedback and self-regenerative mechanisms to continuously improve AI responses.
- **Evaluation Using RAGAS**: Using predefined evaluation metrics to assess the quality and accuracy of AI-generated responses.
- **Self Memory**: Creating a dynamic, evolving memory database to store and reuse relevant past responses for future queries.

1.2.2 Explanation of Approach

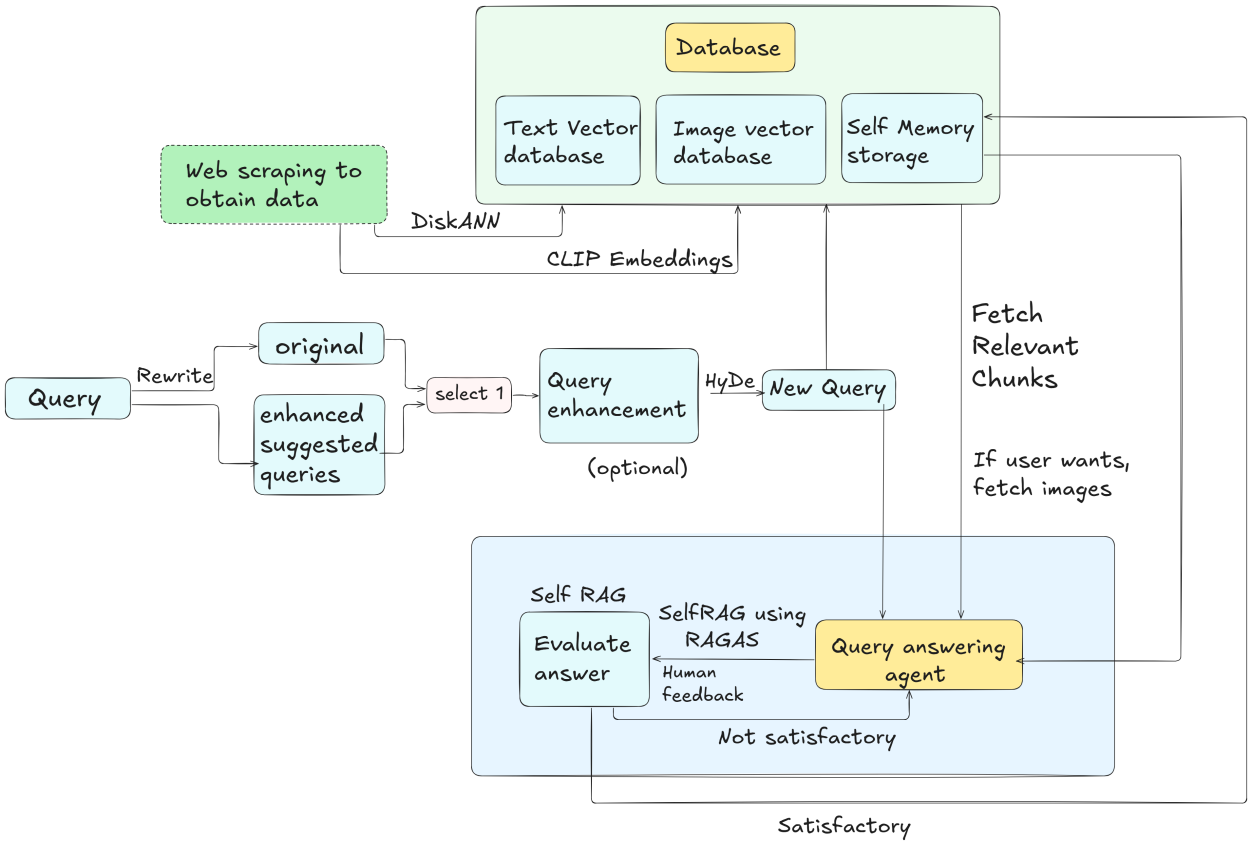
- The system begins with enhancing the user's query through the Query Rewrite and Hyde methods to improve understanding.
- The input data is transformed into structured JSON chunks, and DiskANN is used to cluster these chunks for optimized and faster retrieval.
- Once the AI generates an answer, the system collects user feedback, which influences the regeneration of the answer to improve metrics such as relevance and accuracy.
- The system evaluates responses using the RAGAS evaluation criteria, ensuring high-quality responses.
- A self-memory system stores query-response-context pairs that are used to improve future queries by providing context-aware responses.
- The user interacts with the solution through a clean and efficient Streamlit frontend, while a VS Code extension allows direct integration into the developer workflow.

1.3 Novelty

- **Automated Domain-Specific Corpus Construction:**
  The system uniquely automates corpus generation by scraping official MATLAB and Simulink documentation, including code references and visual assets. This ensures that the knowledge base remains up-to-date, structured, and domain-relevant with minimal manual intervention.
- **User-Guided Query Enhancement via RRR and HyDE:**
  A dual-stage query enhancement pipeline combines Recursive Retrieval-Refinement (RRR) and Hypothetical Document Embedding (HyDE). RRR generates multiple enriched query rewrites, from which the user selects the most relevant, while HyDE deepens semantic alignment. This approach significantly boosts retrieval precision and user control.
- **Efficient Multimodal Retrieval with Structured Chunking and CLIP:**
  Structured semantic chunking based on document layout is paired with DiskANN for fast and scalable retrieval. Additionally, CLIP embeddings enable visual search capabilities, supporting image-based queries related to Simulink diagrams and GUI elements, thus enhancing multimodal troubleshooting.
- **Adaptive Learning through Feedback and Self-Memory:**
  A self-regenerative feedback loop reprocesses poor responses using user critique, while a self-memory layer recalls past interactions to refine future outputs. Combined with real-time RAGAS evaluation metrics, this creates a continuously learning and improving system.

2 System Architecture

2.1 Architecture Diagram



2.2 Detailed Implementation

- **Corpus Acquisition via Automated Web Scraping of MATLAB Documentation**
  The first phase involves constructing the knowledge base by programmatically scraping the official MATLAB and Simulink documentation websites. Using Beautiful Soup 4, relevant pages are crawled and parsed, including troubleshooting guides, function references, block descriptions, and visual assets such as Simulink block diagrams and plots.

- **Intelligent Query Enhancement via RRR and HyDE Techniques**
  The user query is first refined using two methods: Recursive Retrieval-Refinement (RRR), which iteratively rewrites the input to better align with indexed content, and Hypothetical Document Embedding (HyDE), which generates a plausible response and reuses it as an expanded query. This dual-stage enhancement increases the depth and specificity of retrieval, ensuring better semantic matching.
  The RRR method produces multiple contextually enriched query rewrites, and the user is allowed to select the most suitable version before proceeding, giving them greater control over the retrieval focus.

- **Structured Document Chunking and Efficient Vector Search with DiskANN**
  MATLAB documentation is converted into structured JSON and semantically chunked based on logical units like sections and headers. These chunks are indexed using DiskANN (Discrete Approximate Nearest Neighbor), which uses quantization and K-means clustering to reduce redundancy and accelerate nearest neighbor search in high-dimensional space.

- **Multimodal Indexing and Image-Based Retrieval with CLIP Embeddings**
  To improve handling of Simulink and GUI-related queries, visual content (e.g., diagrams, flowcharts) is embedded using CLIP (Contrastive Language–Image Pretraining). These embeddings are stored in the same vector database, allowing the system to retrieve relevant images alongside textual answers when the query requires visual elements.

- **Human-in-the-Loop Feedback System and Self-Regenerative RAG Mechanism**
  Once an answer is generated, users can rate it or provide feedback. If the feedback indicates a poor response, a Self-Regenerative RAG mechanism reprocesses the query by incorporating critique signals. New documents are retrieved, and the answer is regenerated, enabling improvement through user-in-the-loop adaptation.

- **Self-Memory Embedding and Historical Learning**
  A self-memory layer stores previous query-response-feedback tuples in a dynamic vector store. When new queries arrive, the system retrieves and leverages semantically similar past cases, enriching the context and refining the generated solution based on cumulative experience.

- **Quality Assurance using RAGAS Evaluation Metrics**
  The accuracy and reliability of answers are evaluated using RAGAS (Retrieval-Augmented Generation Assessment Score). This includes metrics such as context precision, answer relevance, and faithfulness to source material. Low-scoring responses are flagged for improvement or regeneration.

- **Interactive Troubleshooting Experience via Streamlit Interface and VS Code Plugin**
  The troubleshooting experience is deployed across two platforms:
  - *Streamlit Frontend*: Enables query submission, result viewing, and feedback collection via a web interface.
  - *VS Code Plugin*: Offers in-editor query resolution, showing suggestions and explanations directly within the development environment without disrupting the workflow.

2.3 Issues Faced

The following challenges were encountered during the development and enhancement of the RAG system and its pipeline:

- **Operating System Issues with Streamlit**: On Windows, issues were observed while using Streamlit's page switch methods to navigate between webpages, which hindered the seamless operation of the application.
- **Selecting Evaluation Criteria for the RAG Pipeline**: Several evaluation criteria were found to be unfeasible, as they required ground truth and generated answer data for evaluation. However, the absence of ground truth data rendered these criteria inapplicable.
- **Dataset Acquisition for RAG**: As no pre-existing dataset was provided, the required data had to be manually scraped and organized using the BeautifulSoup library to ensure it was suitable for the RAG system.

2.4 Results and Metrics

- **DiskANN and CLIP Embeddings for Vector Search**: Implemented DiskANN with CLIP embeddings to improve the search time complexity of the RAG system. This combination optimized the efficiency of embedding-based retrieval, resulting in faster and more relevant search results.
- **Caching in Streamlit**: Integrated caching mechanisms within Streamlit to prevent the website from reloading entirely with each interaction, similar to a hot reload. This enhancement reduced latency by more than 100 percent, effectively halving the response time and improving the user experience.
- **RAGAS Evaluation**: Introduced a standardized evaluation metric for the RAG system using the RAGAS framework. This approach ensured consistency and objectivity in assessing the quality of responses generated by the system.

3 Working Prototype

3.1 Streamlit Interface

Our prototype consists of a streamlined and user-centric web interface that facilitates interaction with our system. The following sequence of pictures outline the core functionalities of the application:

- **Login Page (Fig. 1):** The user is first greeted with a simple and intuitive login interface, where they can securely access the system using their credentials.
- **User Analytics Dashboard (Fig. 5):** Upon successful login, the user is redirected to a dashboard that provides insightful analytics and metrics. This panel helps users monitor system usage and performance trends.
- **Query Input Page (Fig. 3):** The next interface allows users to submit their queries. The layout is designed to capture input clearly, ensuring that the user's intent is preserved accurately. At the same time, user can also select/dis-select various parameters such as number of chunks to fetch, HyDE, etc to cater to different needs
- **Answer and Evaluation Page (Fig. 4):** After processing the input, the system displays the generated answer. Additionally, an evaluation matrix is presented to assess the quality and relevance of the response. This includes various parameters such as accuracy, completeness, and response time.

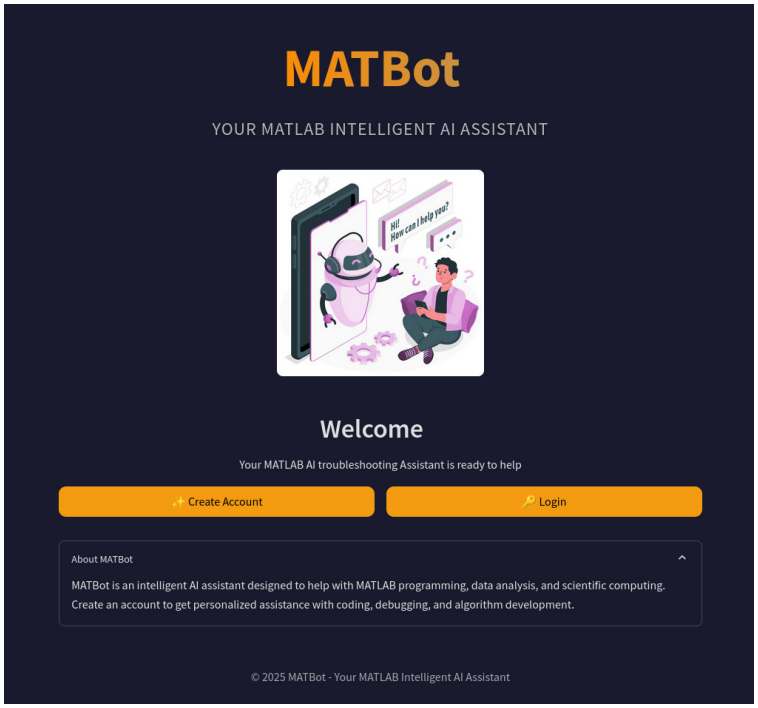


Figure 1: Login Page

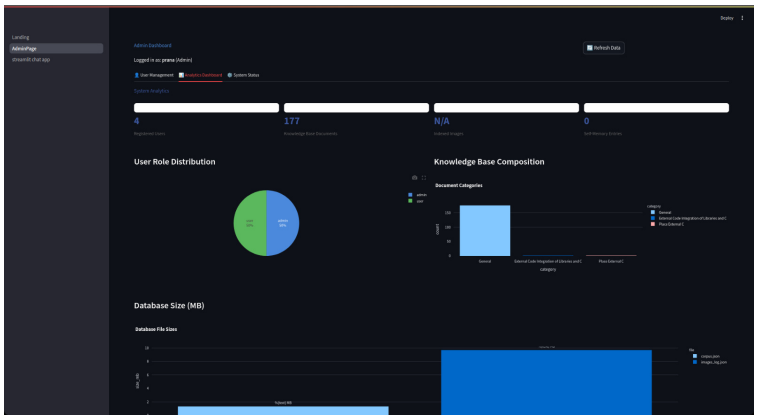


Figure 2: User Analytics Dashboard

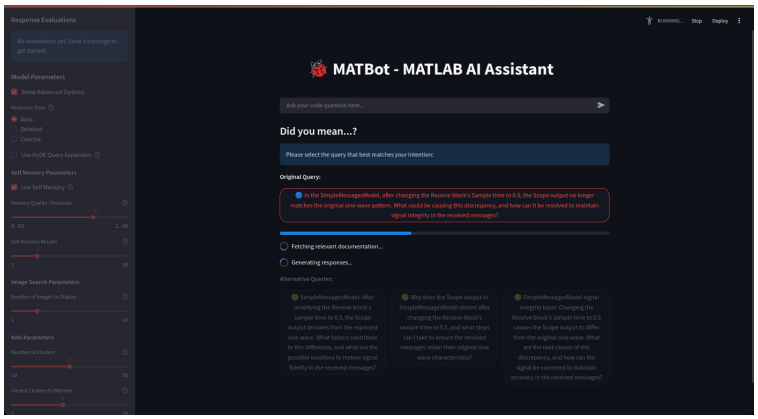


Figure 3: Query Input Page

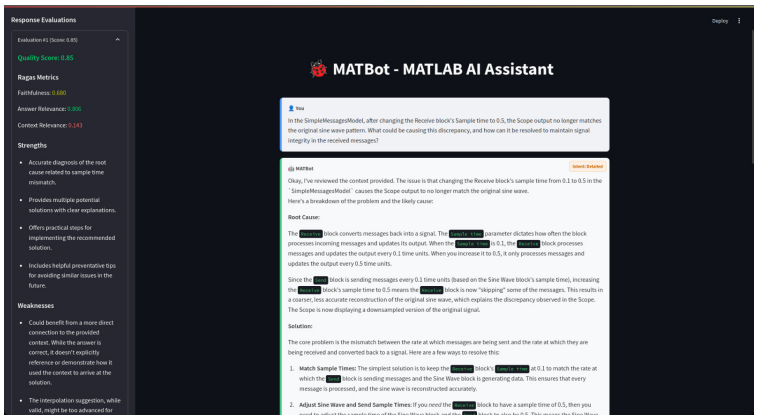


Figure 4: Answer with Evaluation Matrix

3.2 VS Code Extension

We also propose a VS Code Extension that is able to read user code files, answer queries related to the files and modify file content dynamically. The following image shows a demonstration of the same:

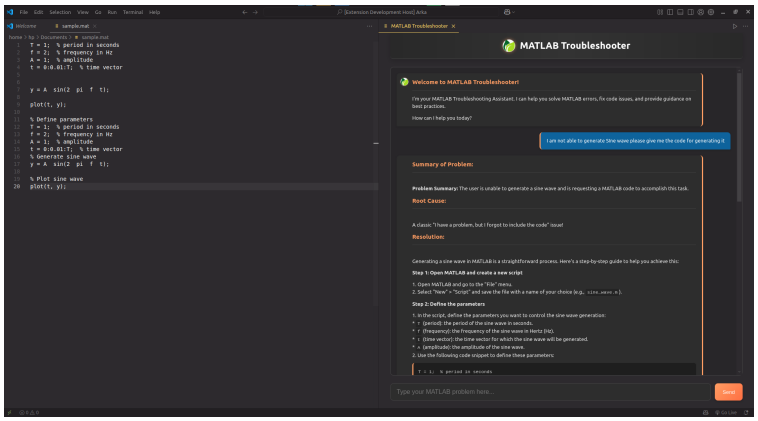


Figure 5: VS Code Extension

4 Future Scope

- **Implementing Graph RAG using DiskANN for Knowledge Representation**
  Future development will focus on integrating a Graph-based Retrieval-Augmented Generation (Graph-RAG) system. This will allow the model to reason not only over text but also over structured interconnections between entities. DiskANN will be used for high-speed approximate nearest-neighbor searches over graph-augmented embeddings. Each query will be dynamically expanded through its relevant graph neighborhood, enabling retrieval that is semantically relevant and structurally contextualized. This system will support multi-hop reasoning and comparative analysis, especially for complex or cross-referenced use cases.

- **Enhancing the VS Code Extension for Compatibility with Evolving Model Architectures**
  The existing Visual Studio Code extension will be upgraded to support seamless integration with updated versions of underlying language models and vector backends. The extension can be enhanced with adaptive model routing, error resilience, and dynamic environment detection, optimizing interaction without manual reconfiguration. This will enable the system to work with the latest reasoning agents and embedding pipelines directly from the IDE.

- **Upgrading the Frontend with an Interactive React-based Interface**
  The current Streamlit-based frontend, while effective for prototyping, will be evolved into a robust, production-grade application using React. A React-based UI will provide finer control over interactive components such as:

  - Live syntax-highlighting for queries and function references
  - Dynamic document viewers for multi-page guides and instructions
  - Responsive query refinement suggestions with ranking previews
  - User-driven feedback panels, visual tracebacks, and session history

- **Incorporating Advanced Evaluation Metrics with Ground Truth Datasets**
  To go beyond traditional evaluation systems like BLEU or RAGAS, we plan to incorporate emerging evaluation frameworks that rely on annotated ground truth. These frameworks include:

  - **FRAMES (Fact-based Retrieval-Aware Metric for Evaluated Summarization)**: Assesses whether generated responses reflect actual structured factual elements from the dataset, particularly useful in summarization tasks.
  - **RaLLe (Retrieval-Aware Language Evaluation)**: A metric for evaluating language generation, focusing on data accuracy, clause preservation, and contextual consistency.
  - **RagElo (Retrieval-Augmented Elo Evaluation)**: Inspired by Elo rating systems, this method performs pairwise comparisons of AI responses using retrieval-grounded evidence, scoring based on consistency and contextual relevance.
  - **MAPPO (Multi-Agent Proximal Policy Optimization) for Evaluation Feedback Looping**: By training policy agents on user feedback and RAG quality metrics, MAPPO enables reward-guided regeneration of responses. This adaptive loop informs fine-tuning and system improvement.

5 Conclusion

This project introduces a robust, agentic RAG-based troubleshooting framework optimized for complex environments like MATLAB. Starting with web-scraped corpus generation, the system enhances queries using Rewrite-Retrieve-Read (RRR) and Hypothetical Document Embedding (HyDE), offering users multiple

refined query options. Semantic chunking ensures meaningful vector representations, indexed efficiently via DiskANN for low-latency retrieval.

Multimodal support is achieved through CLIP embeddings, enabling visual search across diagrams and Simulink components. A feedback-driven Self-RAG loop incorporates user input for iterative refinement, while a self-memory mechanism retains past interactions to enable context-aware assistance in future sessions.

Evaluation is grounded in RAGAS metrics and will expand to advanced frameworks like FRAMES, RaLLe, RagElo, and MAPPO for reward-guided generation. The system is deployed through a Streamlit interface and an upgraded VS Code extension compatible with newer models and runtime environments, supporting seamless in-editor interaction.

Together, these components form a scalable, feedback-adaptive agentic assistant, setting the foundation for future expansion into other technical and analytical domains.

6 Background Study

References

[1] *DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node*: https://suhasjs.github.io/files/diskann_neurips19.pdf

[2] *FreshDiskANN for Similarity Search*: https://arxiv.org/pdf/2105.09613

[3] *Corrective RAG*: https://arxiv.org/pdf/2401.15884

[4] *Adaptive RAG*: https://arxiv.org/abs/2403.14403

[5] *Light RAG*: https://arxiv.org/abs/2410.05779

[6] *HyDE: Enhancing Information Retrieval with Hypothetical Document Embeddings*: https://www.valprovia.com/en/blog/hyde-enhancing-information-retrieval-with-hypothetical-document-embeddings

[7] *Query Rewriting for Retrieval-Augmented Large Language Models*: https://arxiv.org/abs/2305.14283

[8] *Relevant Segment Extraction (RSE)*: https://github.com/NirDiamant/RAG_Techniques/blob/main/all_rag_techniques/relevant_segment_extraction.ipynb

[9] *Improving Retrieval-Augmented Generation through Multi-Agent Reinforcement Learning*: https://arxiv.org/abs/2501.15228

[10] *KILT: a Benchmark for Knowledge Intensive Language Tasks*: https://arxiv.org/abs/2009.02252

[11] *RaLLe: A Framework for Developing and Evaluating Retrieval-Augmented Large Language Models*: https://arxiv.org/abs/2308.10633

[12] *Evaluating RAG-Fusion with RAGElo: an Automated Elo-based Evaluation Framework*: https://arxiv.org/abs/2406.14783

[13] *Performance Evaluation of Multi-Agent Reinforcement Learning Algorithms*: https://www.techscience.com/iasc/v39n2/56498/html

[14] *Retrieval-augmented Text Generation with Self-Memory*: https://arxiv.org/abs/2305.02437

[15] *LangChain Framework for LLM Applications: A Survey*: https://arxiv.org/abs/2305.03983

[16] *Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions*: Xinyi Hou, Yanjie Zhao, Shenao Wang, Haoyu Wang. https://arxiv.org/abs/2503.23278

[17] *AgentBench: Evaluating Foundation Models as Agents*: https://arxiv.org/abs/2308.07308

[18] *Voyager: An Open-Ended Embodied Agent with LLMs*: https://voyager.minedojo.org/

[19] *LlamaIndex: Connecting LLMs to Data*: https://llamaindex.ai/

[20] *ReAct: Synergizing Reasoning and Acting in Language Models*: https://arxiv.org/abs/2210.03629

[21] *Toolformer: Language Models Can Teach Themselves to Use Tools*: https://arxiv.org/abs/2302.04761

[22] *DSPy: A Framework for Programming LLMs Using Declarative Rules*: https://arxiv.org/abs/2402.19152

[23] *AutoEval: Self-Evaluating Agents for Response Quality*: https://arxiv.org/abs/2311.16447

[24] *Gorilla: Large Language Model Connected with Massive APIs*: https://arxiv.org/abs/2305.15334