Inter IIT Tech Meet 13.0 - Pathway PS

Team ID: 41

May 26, 2025

Contents

1 Introduction

The need to build a reliable, adaptable and differentiable pipeline that permits the collection, processing, and production of information beginning from various data sources is addressed by the Agentic Retrieval-Augmented Generation (RAG) system. This solution satisfies the need for context-sensitive, accurate results while making efficient use of resources by using intelligent agents to carry out tasks like context search and dynamic query refinement. The system, which is designed for technical documentation scenarios, incorporates excellent elements such as dynamic thresholding, adaptive retrieval, and personalized re-ranking to support optimal performance across intervals of varying query difficulty. Therefore, the goal of this project is to establish new benchmarks for intelligent and optimal information retrieval systems by implementing such approaches into practice.

2 Uniqueness

2.1 Adaptive Retriever Selection and Hybrid Retrieval Fusion

Chooses and mixes dynamically retrievers according to the query's type through an Intelligent Query Classifier, which improves the context domain overall. We have used reasoning agents to perform context retrieval for us.

2.2 Dynamic Thresholding and Query Refinement

Modifies the cut off values in relation to the confidence levels and decides to further augment context using web search data.

2.3 Personalized Re-ranking

Here also we incorporate personalized ranking where ranking is done according to the query complexity preference and query context.

3 Use-case selection and Novelty

Our main use case targets acquisition of information from technical documentation because technical documents present technical language and context-dependent information, which requires deeper processing. Some more high-stakes technical scenarios can be:

- **Technical Document Summarization**: The system can create short summaries derived from different sections of a technical document and can adjust the depth of the summaries depending on the query input generated by the user.
- **Problem-Solution Queries**: In the case of questions in which someone needs to inquire about troubleshooting, the design or implementation, it can extract specific portions of the technical document and provide a sophisticated response in conjunction with the information in the document.
- **Research and Development Queries**: In connection to questions that are of research type like: what method, what theory, what case the system can find an indicator to theories methods, or case and provide answers backed by technocrat documents.

4 Solution Overview

The system pipeline is designed to handle diverse query complexities through a modular approach:

4.1 GuardRails

Guardrails are integrated at the query stage to filter out gibberish, empty, or inappropriate inputs using rigorously tested pre-trained models. This ensures faulty queries bypass retrieval and LLM processing, saving time and resources while maintaining efficiency.

4.2 Intelligent Query Categorization Engine

The input query is then passed through an Intelligent Query Categorization Engine which categorizes it as trivial, simple, intermediate, or complex.

4.3 Adaptive Retrieval and Re-Ranking

Depending on the query classification the system uses one of the three simple, intermediate or complex retrieval paths which retrieves context differently. The retrieval paths rely on Multi-vector Search, reasoning agents such as the Chain of Thought (CoT), Multi Chain of Thoughts(MCoT), and re-rankers.

4.4 Thresholding Mechanism

Next, the retrieved content is checked again by a thresholding mechanism that will assess the sufficiency and relevance of the content to the process before moving forward. Based

on the relevance, we can decide to get contexts from the Web.

4.5 Output Generation via LLM or Web Search

Finally, with the question and the context ready, we are ready to generate the answer from the LLM. We have also added a feature where the LLM can directly handle non-retrieval based questions such as *Good Morning*, *How are you?* etc.

5 System Architecture

5.1 Flow Chart of the Pipeline

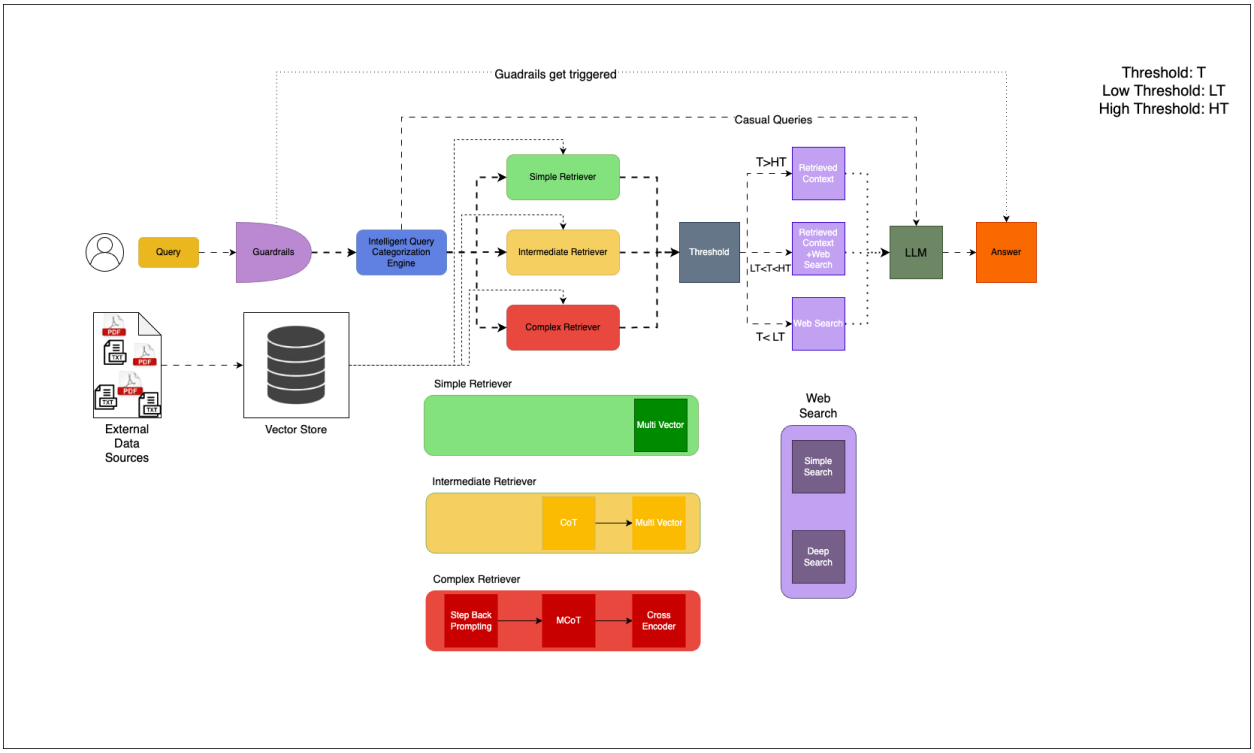

Figure 1: Pipeline Flowchart

5.2 GuardRails

[1] We use pre-trained HuggingFace models to detect gibberish, NSFW content, bypassing retrieval and LLM for flagged queries to save time and tokens. Additional GuardRails at the LLM output were not necessary as the advanced endpoint models already have built-in safeguards.

5.3 The Intelligent Query Categorization Engine, or IQCE

It is a component of the system that has been designed to replace the Mixture of Experts MoE. A Large Language Model (LLM) that has been pre-trained to differentiate between simple, intermediate, and sophisticated queries, is used in the prototype. The IQCE uses an LLM to regulate the dynamic query content in order to adhere to a suitable acquisition technique. In our pipeline, queries vary in complexity. The IQCE framework allows dynamic selection of the appropriate retriever based on the query type: **Simple Retriever, Intermediate Retriever, and Complex Retriever**. It also takes care of trivial(casual) queries.

5.4 Vector Store

[2]The Vector Store was integrated for efficient data retrieval, managing high-dimensional embeddings for fast context retrieval. An abstraction layer, the IndexServer, was implemented to test various indexing strategies in Pathway. After testing, the Usearch Index was found to offer the best balance of speed and relevancy, outperforming options like LSH and hybrid indices. As a result, Usearch was chosen as the default index for its superior retrieval quality.

5.5 Retriever Agents

5.5.1 Simple Retriever

For *simple queries*, the query is directly passed to the Vector Store to fetch relevant information. The retrieved context is then forwarded to the next stage of the pipeline for further processing.

5.5.2 Intermediate Retriever

For *intermediate queries*, to faciliate the the alternate relations between the retrieved documents we use **Chain of Thought** [3] based retriever that divides the query into sub-queries and retrieves the k-top relevant documents; then, the documents are merged and passed to next stage for ranking.

5.5.3 Complex Retriever

For the *complex queries*, to establish the alternate and sub relations between the fetched documents effeciently, we deploy Multi Chain of Thought **MCoT**. To retrieve the global context we have used **Step-Back Prompting** [4], a very simple technique that enables LLMs to abstract specific details into high-level concepts or principles, thus improving multi-step

reasoning and the understanding of complex queries. Later each query is broken down again into simple sub queries that will fetch the similar context from the documents. This MCoT based retriever hence performs repeated and consecutive searches to obtain a wide context, gradually narrowing the search sphere in each subsequent step.

5.6 Re-rankers

Re-rankers interact with the retrieved documents in terms of the complexity of the query. For such simple queries, we deploy a lightweight multi-vector model known as colBERT. Intermediate queries are met by a cross-encoder model known as BGE-m3. For detailed query processing as well we have moved forward with cross-encoder model BGE-m3 after testing with few other different re-rerankers such as LLM reranker.

5.7 Thresholder

Thresholder [5] checks if the retrieved contexts match the user's question, assigning one of three scores: relevant, ambiguous, or irrelevant. If the score is above the higher threshold, those are treated as relevant, and the retrieved context is used directly. If the score falls between the lower and higher thresholds, those are treated as ambiguous, and the retrieved context is used along with a web search on the query. If the score is below the lower threshold, those are considered as irrelevant, and it proceeds directly to a web search. The goal is to quickly filter out clearly wrong matches while retaining uncertain ones for further review.

5.8 Web Agent

[3]The Adaptive Web Search module supplements the RAG system by integrating external information when the internal database is insufficient. A simple agent dynamically selects between two strategies:

5.8.1 SimpleSearch

For straightforward queries, this strategy retrieves up to five results via direct API calls, ensuring fast and efficient responses.

5.8.2 DeepSearch

For complex queries, this approach breaks the query into sub-queries, retrieving up to three results for each, enabling comprehensive context coverage.

To ensure uninterrupted operation, the module incorporates error management, automatically switching to backup APIs if primary sources fail, maintaining reliability and seamless functionality.

6 Results and Metrics

Here we present the comparison of re-rankers across various query types and metrics.

Query Type	**colBERT**		**BGE-m3**	
	NDCG@10	**MRR@10**	**NDCG@10**	**MRR@10**
Numeric	0.38	0.33	0.39	0.35
Description	0.37	0.32	0.38	0.34
Entity	0.40	0.35	0.40	0.36
Person	0.42	0.36	0.42	0.36
Location	0.54	0.48	0.55	0.50

Table 1: Performance comparison of colBERT and BGE-m3 as re-rankers across query types for MS MARCO dataset.

	colBERT Overall		**BGE-m3 Overall**	
Metric	**NDCG@10**	**MRR@10**	**NDCG@10**	**MRR@10**
Score	0.42	0.37	0.43	0.38

Table 2: Overall performance of colBERT and BGE-m3 as re-rankers for MS MARCO dataset.

We validated our RAG model by applying the RAGAS framework on five different test scenarios for context precision, context recall, faithfulness, and answer relevancy. The results prove the efficiency of the method considering context-based precision, recall, and the relevance of answers provided (the mean values approaches 1). More modest are the numbers concerning the faithfulness: they range from 0.75 to 0.81, and correlate to the other figures; thus, there is the potential for increasing faithfulness of the recitation. In conclusion, the model reveals high compatibility and appropriateness of the context.

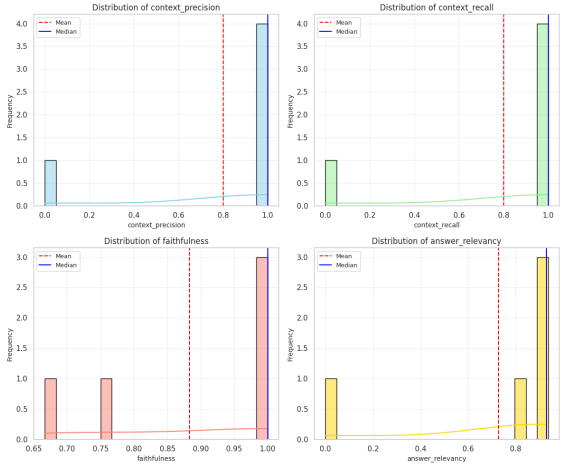

Figure 2: Results for Test Cases

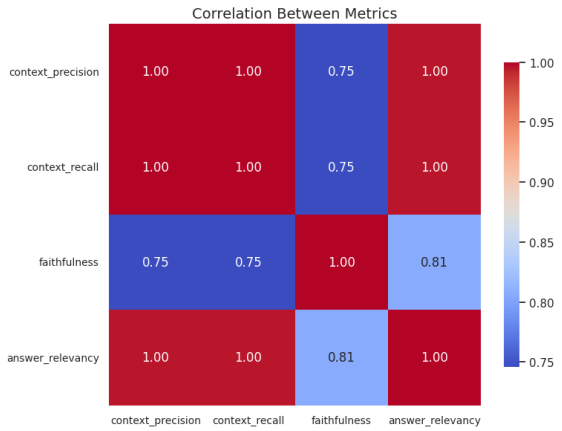

Figure 3: Correlation Matrix

7 Challenges faced and Solutions

7.1 Handling different types of Queries

The main problem we are tackling in our solution is handling different queries. Queries could have varying amounts of contextual depths, such as 'How is the weather today' to 'Fetch me all of the documents where this section of the constitution has been applied'. Thus, we must classify them based on contextual richness, fetch varying amounts of context for each query, and augment with Web Search data if needed to answer properly.

7.2 Extensive Context Retrieval

There are two levels of Context Retrieval based on the source:

- Vector Database Retrieval
- Web Retrieval

One key challenge was implementing dynamic retrieval strategies that incorporate local and global relationships without rebuilding structures like graphs for each new document. These advanced retrieval methods were needed selectively, requiring the development of an Intelligent Query Classification Engine (IQCE) to decide which queries needed such strategies. While simple queries could directly access the database, intermediate ones required additional context derived through Chain-of-Thought (CoT) reasoning, which helped establish local relationships.

For complex queries, we implemented a Multi Chain-of-Thought (MCoT) framework. It generated Alternate Questions for global context and sub-queries for local relations,

ensuring comprehensive retrieval. However, integrating and orchestrating these layers dynamically while maintaining accuracy was challenging.

Parallel implementation of Simple, Intermediate, and Complex Context Retrievers also introduced difficulties. Each retriever (e.g., CoT, MCoT) interpreted queries differently, leading to inconsistencies and requiring extensive fine-tuning to maintain coherence. Additionally, while the Tree of Thoughts (ToT) architecture was initially considered for complex queries, it proved infeasible due to its high retrieval time. Replacing it with a tailored MCoT framework improved efficiency but required significant pipeline modifications.

These challenges demanded innovative solutions and rigorous optimizations to achieve a scalable and accurate retrieval system.

7.3 Data Handling and Vector Store Integration:

Integrating the Vector Store for efficient data retrieval involved challenges with data formatting and aligning vectorization with database requirements. Initially we wanted to test out different indexing strategies and pick the best one for our case. Pathway had various indices (lsh, usearch, bruteforce and vector document) but it seemed like the version that we were using (pathway==0.15.3) was only using usearch knn by default and we didn't have an option of choosing. So we made an abstraction out of the vector store (called IndexServer) which allowed us to use our index of choice and after rigorous testing we found that lsh and usearch had similar speeds and among them usearch retrievals had better relevancy. We have also tried using a Hybrid Index with Usearch and BM25 but the retrieval speeds were too slow. So we have finally decided to keep Usearch Index.

7.4 Thresholding for Effective Routing:

Setting thresholds for activating the LLM or Web Search retrievers is complex, requiring extensive testing to select the right retriever based on query complexity. Dynamic thresholds remain a technical challenge but are becoming more adaptable through continuous testing. We plan to improve the thresholding mechanism through further testing and calibration, aiming for real-time selection of the most suitable retriever based on query complexity.

7.5 GuardRails

We also encountered several hurdles while integrating GuardRails. Many of the models available intended to address specific issues were not sufficiently reliable. This required us to rigorously test multiple options to identify the most effective one before successfully incorporating it into our pipeline.

8 Resilience to Error Handling

Ensuring robustness within the pipeline is crucial, especially in error-prone areas. The web search component, in particular, is a potential point of failure if the primary API encounters issues. To address this, a fallback mechanism has been implemented, integrating four web search APIs. If the primary API fails, the system seamlessly switches to the next available API, ensuring uninterrupted retrieval. These strategies collectively enhance the system's resilience, minimizing disruptions and maintaining consistent performance.

9 User Interface

The **Pathway Chatbot** interface, built with **Streamlit**, provides a user-friendly experience.

- **Query Input**: Users can type their questions in the text box, and the chatbot generates responses.
- **Feedback Options**: Users can like or dislike responses, with feedback statistics displayed in a summary section.
- **Report an Issue**: Users can report issues or provide feedback by selecting categories such as "Feature Request" and describing the details.
- **Developer Notifications**: Submitted reports are emailed to the developer, including issue details and like/dislike statistics.

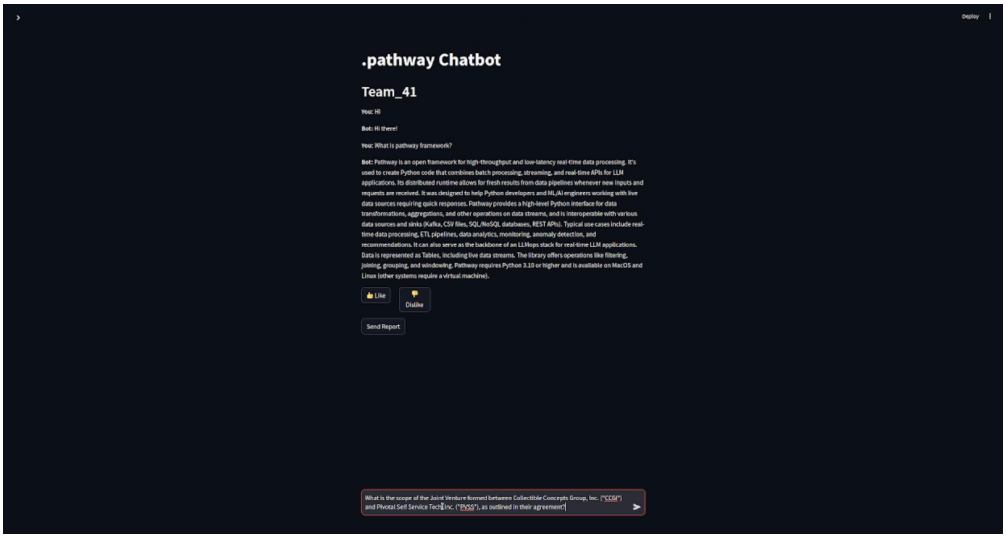

Figure 4: User Interface

This intuitive design enhances user engagement and supports continuous system improvement.

10 Responsible AI Practices

We have integrated GuardRails into our RAG pipeline, by placing it at the most critical stage: processing user queries before context retrieval. This GuardRail acts as a filter to identify and handle potential issues such as offensive queries, gibberish, or empty inputs. It ensures that, in such cases, the pipeline provides an immediate and appropriate response without proceeding to retrieve context, thereby directly delivering an answer.

11 Lessons Learned

- In this regard, the concept of adaptive retrieval techniques such as IQCE is particularly crucial to handle the complexity of the range of queries.
- When system failure occurs, it can be reduced by modifying the threshold level and includes additional fail-safe methods to guarantee that system weaknesses are addressed and system performance is enhanced.
- Within the principles of this strategy, robust input filtering takes the form of GuardRails, which improve dependability and eliminate inefficiencies.
- Positive user feedback facilitates case usability and improves system adaptation.

12 Conclusion

The work was done during the project to deliver a dynamic agentic retrieval-augmented generation framework to solve different queries intricacies and technical documentation-related tasks. Key achievements include:

- Adapting approach by creating parts that may be combined in varying structures: GuardRails for approved info retrieval and their integration for qualified consumption, IQCE, and MCoT.
- Developing efficient error handling capability through fallback class APIs and other LLM back-ups to support effective features.
- As an example of how this system can be extended to multiple domains and provide accurate and domain-specific results.
- highlighting the various works in integrating the methodologies and some of that includes the personalized re-ranking and the written hybrid retrieval fusion.

References

[1] Towards Data Science. Safeguarding llms with guardrails.

[2] Pathway.com. Pathway's vectorstore.

[3] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2023.

[4] J. Smith and A. Doe. Take a step back: Evoking reasoning via abstraction in large language models. *Journal of Important Studies*, 2021.

[5] Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. Corrective retrieval augmented generation. *arXiv preprint arXiv:2401.15884*, 2024.