Mid Evaluation Report

Dynamic Agentic RAG with Pathway

Team 34

I. INTRODUCTION

Despite their excellent semantic understanding and reasoning capabilities, large language models (LLMs) often hallucinate when handling domain-specific or private user queries. These queries often require additional contextual information beyond the inherent knowledge of the model. Retrieval-Augmented Generation (RAG)[1] is an approach that helps to augment an LLM with relevant context with the help of which user queries from a variety of backgrounds can be answered, such as private company laws, financial reports, etc. Such complex queries require a dynamic RAG system, which can perform multi-step retrieval along with invoking LLM's reasoning ability to address the user query coherently. The system should be able to dynamically understand when to retrieve and when to analyze the retrieved information. Moreover, the queries are not always limited to answering from a given context. They may require different tools, such as a calculator for performing algebraic operations on performed reasoning or a data analysis tool for generating charts based on the retrieved information, etc., especially when reasoning over financial reports, necessitating the integration of multiple agents with a robust RAG system to handle a diverse set of tasks effectively.

We are building a dynamic agentic RAG system capable of performing the above-mentioned tasks and beyond in specific domains such as legal and finance. Such domains often involve long, unstructured reports with high token counts, which pose challenges for traditional retrieval methods. Our current system is equipped to perform coherent retrieval and reasoning on long documents, such as hundred-page PDFs while supporting tool-based reasoning and further agentic behavior. We also build mechanisms for managing multiple tool failures and incorporate human-in-the-loop to ensure robust performance and accurate results.

We first briefly introduce our system architecture and it's key components. We then detail each component, explaining its functionality, purpose, and the specific challenges it addresses. Finally, we present the evaluation of each component, justifying the design choice and present limitations in the current system and future plans.

II. SYSTEM ARCHITECTURE

Our system employs a multi-tool, multi-agent architecture (Fig 1), where a lead supervisor coordinates the use of various agents and tools necessary to effectively address user queries. The supervisor is currently a rule-based agent that routes the query to different components, evaluating their responses and invoking additional agents or tools as required to handle failures. Our system comprises of following key components:

1. Supervisor: Currently a static rule based agent, it routes the query to required components and perform orchestration between multiple agents for handling any failures and final response generation.

2. RAG Agent: Our RAG agent efficiently handles both small and large documents. It starts with document-level retrieval using JINA embeddings[2] to identify relevant pages. The Unstructured library extracts key elements like tables and headers, which are summarized for enhanced reasoning. We then build a vector index using RAPTOR [3], a tree-based structure that captures text order. Our query engine uses this index and employs interleaving to dynamically switch between retrieval and reasoning, ensuring coherent responses.

3. Code & Reasoning Agent: This is an LLM-based agent that performs the decomposition of queries into smaller components and generates step-by-step Python function calls to invoke the required tools and agents for each subtask. Our system treats agents as tools, and C&R agents handle the planning and coordination of these agents and tools. This step-by-step function calls allow for handling tool failures robustly without affecting subsequent calls.

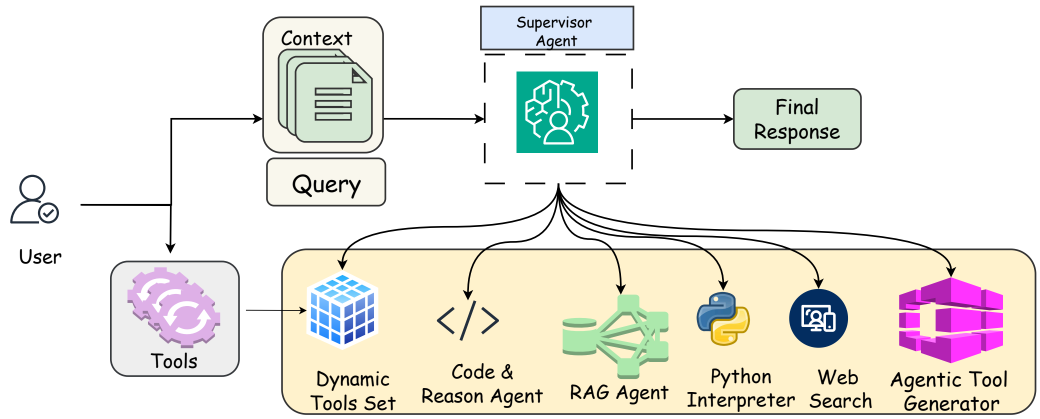


Fig. 1: Illustration of System Architecture comprising key components: Agents, Tools, and Inventory.

4. Dynamic Tool Set: A dynamic tool inventory is crucial for developing agent system. Specifying a set of tools is essential, considering that these tools are intended for agent interactions. Our system utilizes the scalable and standardized Llama-Index framework, establishing a generic interface between tools and the agents. This framework enables effortless integration, allowing users to add new tools efficiently.

5. Agentic Tool Generator: It is an LLM agent for building real-time agents to mitigate tool requirements. Sometimes, the user-provided tool might be corrupt, resulting in internal API failure with the absence of alternative tools. In such scenarios, the supervisor can invoke a generator to generate a meta-prompt, which then can be provided to an LLM resulting in an agent that can mitigate the required tool behavior.

III. RAG AGENT

Large-document retrieval and reasoning is challenging for RAG systems, as most models are trained on smaller text (e.g., 100-word paragraphs), making extensive documents like PDFs difficult to handle. Recent methods [3] use a bottom-up approach, clustering and summarizing segments to create a hierarchical tree structure. This approach supports complex queries and multi-step reasoning but has exponential time complexity, making it unsuitable for PDFs with `num_pages` > 100 [4], as shown in Fig [2].

Our proposed RAG Agent as shown in Fig [3] decides at each stage either to use a reasoning or retrieval to solve a given query. If it decides on retrieval it uses a 2-stage retrieval process. First, for a query Q, we retrieve the top K pages based on similarity scores between the query Q and page-level embeddings P_i . These top K pages are then sent to the RAPTOR module, which clusters the context and forms a hierarchical structure to enable fast and effective retrieval. Alternatively if it decides on reasoning, it uses the current context to reason out the answer. This interleaved approach not only enhances retrieval efficiency but also ensures that reasoning is applied only when necessary, reducing redundant retrieval calls The full pipeline for this RAG agent is detailed in the following sections.

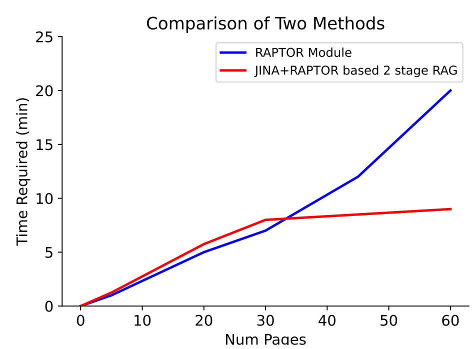


Fig. 2: Time consumption for context retrieval with and without Page level retrieval

A. Page Level Retrieval

Traditional approaches embed a query Q and PDF pages into single-vector embeddings, calculating similarity scores between the query embedding q and each page embedding P_i to retrieve the top K pages. However, short context windows in embedding models lead to information loss in large documents, and the embeddings are not optimized for retrieval tasks. Methods like LongRAG [5] aggregate documents into large chunks, but this requires LLMs with extended context windows and is inefficient for multi-hop question answering, which needs smaller, varied chunks. Structured Hierarchical Retrieval uses metadata summaries for semantic search, but proves inefficient with

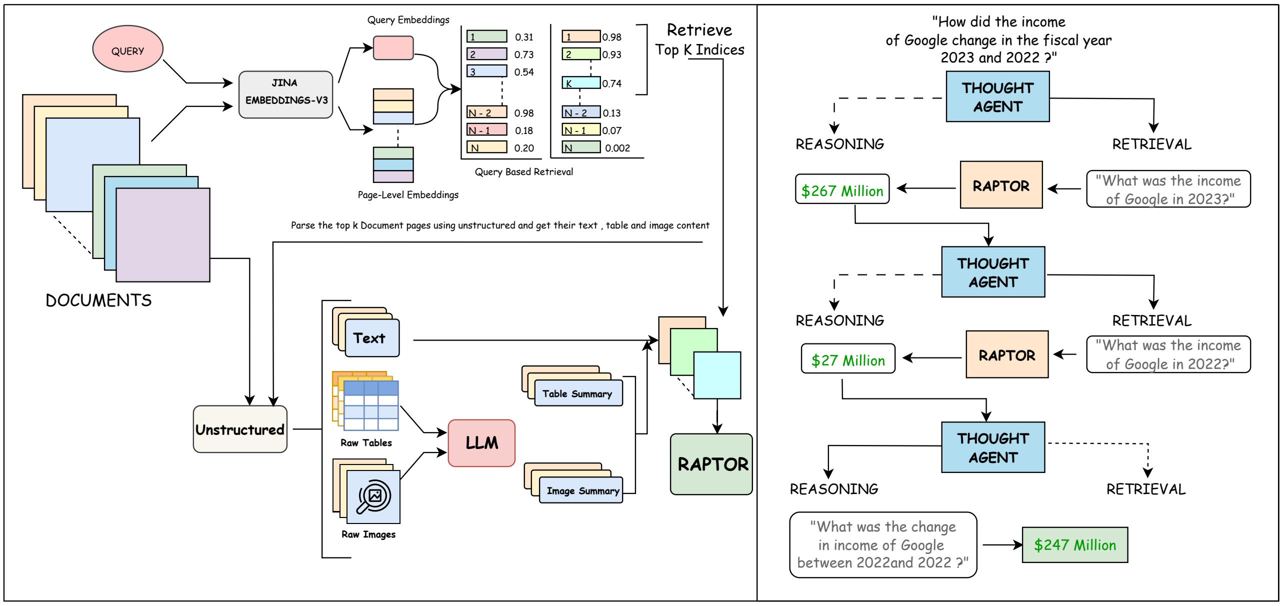


Fig. 3: The JINA Embedding model retrieves the top K similar pages, parses and summarizes text and tables, and recombines them pagewise to create the RAPTOR context. The thought agent then dynamically decides whether to retrieve or reason through a step-by-step process when given a query.

large documents. To tackle the above problems , we use **Jina Embeddings-v3**, which has two key properties that make it suitable for our task. First, it is optimized for long-document retrieval using ALiBi[6], an alternative to traditional positional encoding that prioritizes recent tokens, enabling effective handling of lengthy documents. Second, it employs 5 task-specific LoRA adaptors[7] to optimize embeddings for various tasks, including `retrieval.query` and `retrieval.passage`. Formally, given an initial query Q and a long document D with N pages, we generate query embeddings Q_e and page-level embeddings P_{ie} using the Jina Embed model. Page embeddings are stored in a FAISS vector store for efficient retrieval. Using cosine similarity, we retrieve the top K pages based on Q_e.

B. *Preprocessing of Retrieved Documents*

Throughout our experiments, we observed that PDFs, especially in the financial domain, contain valuable information in tabular format. To ensure accurate extraction, we use **Unstructured.io** to parse the retrieved pages into images, plain text, and tables (as markdown HTML files), each with associated metadata (page number). This metadata facilitates the recombination of page-wise information post-preprocessing. The extracted tabular data in HTML format is then passed to an LLM, instructed to summarize the data for retrieval tasks. Finally, the text components, image summaries, and table summaries are recombined page-wise, enhancing the overall content with insights from the tables and images which helps in proper LLM understanding of the given context. This preprocessed document set is sent to the second stage for final context retrieval.

C. *Context Retrieval*

RAPTOR[3] is a bottom-up indexing approach that uses clustering and LLM-based summarization to form a hierarchical structure. This makes **RAPTOR** a natural choice in financial and legal domain document retrieval over Vanilla RAG and other similarity based retrieval techniques, as it efficiently organizes large volumes of complex data into digestible summaries, enabling faster and more accurate responses to specific queries. But it is impractical for large documents $>$100 pages due to LLM based summarization. However our use JINA Embed model for document retrieval, ensures relevant context is captured within the top 30 pages which can be efficiently handled by RAPTOR. This enables RAPTOR to build a hierarchical tree structure in under 4 minutes. The process involves segmenting the document into text chunks, embedding them, clustering the embeddings, and summarizing each cluster with an LLM. The generated summaries form a tree structure, with higher nodes providing more abstract summaries, aiding in domain-specific query answering.

D. Interleaved Reasoning

A robust RAG system should dynamically decide when to retrieve information and when to analyze it for further reasoning. Since different subquestions may require retrieving distinct paragraphs, a step-by-step retrieval process becomes essential as shown in Fig 3. To address this need, we developed a unique RAG reasoning paradigm called Interleaved Reasoning, where the LLM acts as a decision-maker at each step, seamlessly switching between retrieval and reasoning based on the prior history of retrieved information and reasoning performed. The key features of our Interleaving Paradigm are:

1) Dynamic Decision-Making: At each step, our interleaving agent decides whether to generate a retrieval thought to gather additional context or a reasoning thought to infer answers from the existing information. This approach reduces reliance on external context by optimizing retrieval calls, making them only when necessary. As a result, the agent produces a more coherent and efficient response to user queries.

2) Transforming Failed Retrievals: Based on the history of prior steps, if a retrieval attempt fails, the interleaving agent transform the retrieval thought to successfully obtain the required information. This transformation demonstrates the inherent self-correction capability of our RAG agent, enabled by its interleaving design. Unlike other recent approaches that rely on separate critic modules or query-transforming agents, our system dynamically adjusts in real-time, optimizing retrieval without external intervention.

IV. CODE & REASONING AGENT

In real-world scenarios, RAG systems often need to collaborate with various tools. For example, after retrieving data from a company filing, one may need to perform calculations or record legal responses in a text file. To automate this process, an agent is required to determine the correct order of tool execution, make appropriate tool calls, and handle failures. This is where the C&R agent comes in. It interacts with a set of tools, including both user-provided and general-purpose tools from LlamaIndex. The C&R agent decomposes the query, directs tasks to the RAG agent, and assesses whether specific tools should be invoked based on the RAG response. Key components of the agent include:

A. Chain of Function Call

The Code & Reasoning agent uses step-by-step Python function calls to execute tool and agent reasoning. Each step provides the previous history of Python function calls and corresponding responses, the available set of tools, and the original user query. The agent then decides the next call, which is then executed using an interpreter, resulting in the API response. This process continues iteratively until the user query is fully addressed. While this approach shares similarities with ReAct [8], it is more token and cost-efficient due to its reliance on code-driven planning. Fig 4 demonstrates how the chain of function calling works.

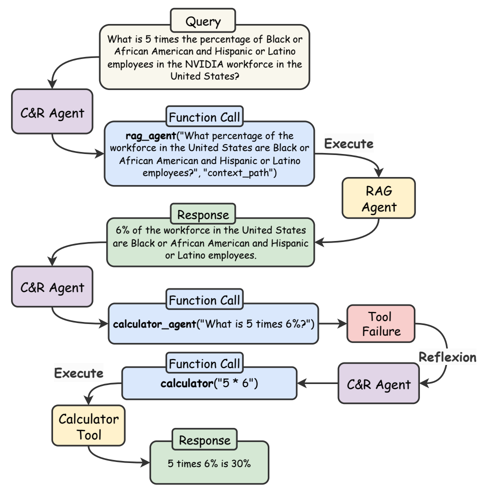


Fig. 4: Chain of Function Calling used by C&R agent for tool reasoning

B. Tool Failure & Reflexion

Sometimes, the APIs can contain internal errors, resulting in a failed response, or the LLM can fail to provide the correct set of inputs, resulting in an error. In such cases, the agent should be capable of handling these errors by invoking alternative APIs or taking corrective actions to resolve the issue and ensure the task is completed successfully. While simple tool failures such as incorrect API input by LLM are majorly focused and can easily be refined using error messages, other silent tool failure scenarios are mentioned in [9]. Building on this work, we additionally account for internal API failures, which are critical in real-world applications. We categorize these tool failures into two main categories based on their execution:

1. Failed function calls: These occur when a generated function call fails to execute, either due to (i) *incorrect python syntax* or (ii) *internal API failure*. The first issue can typically be resolved using code reflexion, which corrects the generated Python function call using the error message. However, the second issue, internal API failure is more permanent one and often cannot be resolved without user intervention. In such cases, the supervisor identifies and discards the faulty APIs, and the C&R agent performs API reflextion, selecting an alternative available API to ensure continuity in the process. Here the system uses failure callbacks to decide the nature of reflexion required as shown in Appendix B1.

2. Executed function calls: These are silent tool failures, where APIs are executed but produce incorrect results due to (i) *incorrect input arguments* or (ii) *incorrect API output*.As illustrated in Appendix B2, in the case of incorrect input arguments, the data type may be correct, but the argument values are inaccurate. This differs from the previous category, where incorrect data types cause failures. These silent failures require a critic agent to evaluate the input and output of the API calls. Based on the critic's feedback, the system identifies the failure. For input argument errors, the failure can be corrected using code reflextion guided by the critic's feedback. On the other hand, incorrect API output usually stems from poor API implementation, which can be addressed using API reflextion, explained in the previous category.

C. Tool Enhancement

While the proposed reflextion policies effectively manage tool failures, there are scenarios where the available toolset may not contain the necessary tools for a user's query, or where all relevant tools are corrupt. We propose two methods to handle such cases as shown in Fig 5:

1. Human-in-the-Loop: In scenarios where the toolset is insufficient, incorporating human-in-the-loop interaction is a practical approach. Here, the supervisor prompts the user to provide the relevant tools needed to address the query.

2. Agentic Tool Generation: If the user is unable to supply the necessary tools, the supervisor agent can invoke an agentic tool generator, which creates agents that mimic the required tool behavior in real time. Recent works, such as [10], have demonstrated methods for automating the design of various agents. Building on these approaches, our system incorporates an agentic tool generator to dynamically design tools that fulfill the required tasks.

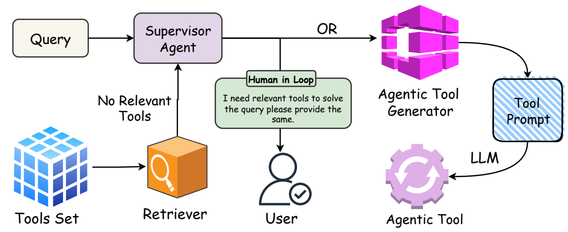


Fig. 5: Tool Enhancement using Human-in-loop and Agentic Tool Generator

V. EVALUATION

A. Experiment Setting

1) Dataset & Queries: For both RAG and Reasoning evaluation, we focus on the FinanceBench and Finance 10k datasets from the financial domain. Finance10k questions are fact-based, requiring "needle in the haystack" type retrieval, while FinanceBench focuses on complex tabular reasoning. For the legal domain, we use the Open Australian Legal Q&A. However, these datasets only require context retrieval from a single page. This limits their generalizability to multiHop question answering which requires context from several pages. To the best of our knowledge, there is no publicly available multihop-based retrieval dataset suited for our specific use case, due to which we curate our own Finance-MultiHop and CUAD Legal-MultiHop queries based on FinanceBench and CUAD[11] to handle Multi-Hop questions across multiple pages. Details about the datasets used have been discussed in Appendix A.

2) LLM & APIs: We use the open-source LLaMa family of models as the foundation for our agents and experiments. Specifically, we utilize LLaMa 3.1-70B and conduct our experiments through Groq and TogetherAI APIs. Using an intermediate open-source 70B model adds to the robustness of our system without depending on more expensive, advanced closed-source LLMs.

B. RAG Evaluation

1) Context Retrieval: We use our self-curated CUAD MultiHop Dataset to evaluate the retrieval performance of traditional RAG techniques against the RAPTOR module. As shown in Table II, the RAPTOR module outperforms all other baselines in terms of query-based retrieval quality. However, experiments also demonstrate (see Figure X) that the time taken by the RAPTOR module increases exponentially with the token count.

2) Page Level Retrieval: Due to RAPTOR's exponential scaling, we many evaluated doc-level retrieval methods to pre-select pages for RAPTOR processing, tested on finance datasets spanning multiple pages. As shown in Table I, Jina Embeddings achieved the highest MRR and time efficiency. Table III details Jina Embedding-v3's performance on FinanceBench, Finance 10k, and Finance MultiHop. Notably, Jina

Method	MRR	Time (for 150-200 page doc)	Documents Considered
JINA CLIP	0.20	8 min	Top 30
JINA Embeddings	**0.31**	**2 min**	Top 30
JINA Embeddings+LLM Reranker	0.247	8 min	Top 15
JINA Embeddings + JINA Reranker	0.23	6 min	Top 15
LONGRAG	0.18	3 min	Top 30
Subdoc Retrieval	-	>50 mins per document	Taking too long. Not feasible for large PDFs
Hierarchical Retrieval	-	95 mins for a 170-page PDF	MRR calculation was infeasible for large PDFs.

TABLE I: Performance Comparison of Various Retrieval Methods on FinanceBench Dataset

Method	MRR	Time (s)
Vanilla RAG	0.65	26
Vanilla RAG + JINA Reranker	0.8	30
Vanilla RAG + LLM Reranker	0.78	75
RAPTOR	**0.81**	130
Structured Hierarchical	-	>300

TABLE II: Comparison of Methods Based on MRR and Time Efficiency on CUAD MultiHop Dataset

performs better on Finance MultiHop and Finance 10k (fact-based queries) than on FinanceBench, which includes more inference-demanding, math-based questions.

3) Reasoning Technique: We experiment with three key reasoning techniques in the LLM domain as shown in Table IV. The first is **Chain of Thought (COT) reasoning**[12], where complex problems are broken down into sequential steps, enabling LLMs to approach each part systematically for greater accuracy. The second technique, **Tree of Clarifications (TOC) reasoning**[13], treats each question as a node that branches into sub-questions, each addressing a specific aspect of the main question. This recursive process continues until no further sub-questions are needed. Lastly, we apply the **Interleaving approach**, which guides the LLM to alternate between reasoning and retrieval as needed to reach the answer. Appendix C also shows a demonstration of two of the reasoning techniques given the same query.

Datasets used	MRR	Top K Docs
FinanceBench	0.31	30
Finance Multihop	0.687	25
Finance_10k	0.57	30

TABLE III: Performance of Jina Embeddings-v3 on Finance datasets.

C. Tool Reasoning Eval

To enhance tool robustness, we expanded our curated queries to include tools like a calculator and data analysis tools, allowing us to evaluate the Code & Reasoning agent. Additionally, we employed an API simulator agent that generates faulty APIs based on queries, enabling testing for callback failures and their resolution. Given its similarity to ReAct, we compared both approaches in terms of token count and cost efficiency, as they deliver comparable accuracy. Our results on sample queries show that C&R significantly outperforms ReAct in token usage, making C&R a more cost-effective tool reasoning method. Figure 6 illustrates this comparison, with additional details provided in Appendix D.

Reasoning Technique	Accuracy
Interleaving	**0.80**
Tree Of Clarification	0.35
Chain Of Thought	0.5

TABLE IV: Performance Comparison of Reasoning Techniques on FinanceBench

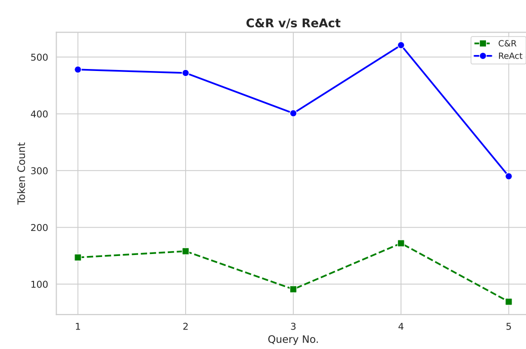


Fig. 6: Token Usage Comparison of Tool Reasoning Agents

VI. Challenges & Future Plan

In this section, we outline some of the challenges we faced by our current system, along with some additional approaches we explored and may revisit in the next stage. We also provide insights into additional features, code integration, and other improvements we plan to incorporate to enhance our current system.

A. Challenges Faced & Limitations

1) Knowledge Graph based Retrieval: Initially, we aimed to adopt a hybrid retrieval module combining vector and KG search, with the supervisor dynamically deciding the appropriate retrieval method based on the user query. The goal of using a knowledge graph was to transform unstructured text into a structured format, making it more suitable for factual and tabular question answering. Recent works, such as [14], have proposed reasoning approaches with KGs that could align well within our Interleaving paradigm.

However, generating good quality KGs from long unstructured text is a complex task, and using LLMs resulted in poor graphs as shown in Appendix E1. We further utilized Llama index's KG query engine, but the graphs were not convincing. We believe if we can tackle this limitation, our hybrid approach will add robustness to our current system.

2) Web surfer for additional Context: During our experimentation [E2], we observed that specific queries in FinanceBench required additional context, such as specific formulae and an understanding of finance jargon and terminology, to solve the questions successfully. These details were neither inherently captured by the LLM nor present in the reports. We can utilize a web surfer to retrieve such necessary in-domain information to address this. This could be seamlessly integrated into our interleaving agent, which can choose to call web agents when needed.

B. Future Plan

1) Conversation Module: We plan to extend the system include a conversation module between supervisor and user to get additional human-in-loop support beyond tool request for information such as further clarification over asked questions, feedback on responses, multi-conversation question answering, etc. This module will also include a memory bank to store previously provided information or asked queries, allowing for easy retrieval without the need for extensive reasoning or retrieval steps.

2) Memory module for Retrieval Optimization: Current LLM agent systems focus primarily on memory modules, storing the history of interactions between the agent and user for future reference. We aim to extend this concept to enhance the RAG retrieval process. Specifically, we propose tagging previous retrieval queries and their corresponding chunks within a graph. Each chunk is associated with a relevant query, forming connections to other related chunks in the graph based on both text and query similarities. If the RAG agent has already answered a subquery, instead of performing a complete retrieval of the entire document, the system can directly access the relevant chunks from the tagged graph. This method significantly improves retrieval efficiency by eliminating redundant full searches, streamlining the process, and introducing a more novel approach to handling queries within the retrieval system.

3) Guardrails: To ensure safe and reliable LLM outputs, we plan to integrate guardrails within both the retrieval and reasoning modules. Using frameworks like Guardrails AI with its 60+ validators, which filter risks such as toxic language, bias, and privacy concerns. NVIDIA's NeMo Guardrails [15] provide programmable constraints for safe dialogue paths, while TruLens enable real-time monitoring and performance evaluation. frameworks will enhance content quality and maintain ethical standards.

4) Integration with Pathway: After building a robust long-document retrieval system, our next step is to integrate retrieval and reasoning (interleaving and C&R) with Pathway. We aim to incorporate our entire RAG pipeline within Pathway, enabling real-time synchronization, detailed metadata handling, and seamless data flow. This integration will include utilizing Pathway VectorStore and enhancing the BaseRAGQuestionAnswerer class to align with our current system.

5) UI Interface: We are planning to develop a web-based user interface that allows users to upload files and ask questions related to the content. The UI will feature a chat interface to store the entire conversation history. Users can easily reference past interactions, and the system will respond to queries by retrieving relevant information from the uploaded files. In addition to providing the content, the user can quickly provide access to a suite of tools for the system through the interface, enhancing agent-tool interaction and facilitating complex queries.

REFERENCES

[1] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459–9474, 2020.

[2] S. Sturua, I. Mohr, M. K. Akram, M. Günther, B. Wang, M. Krimmel, F. Wang, G. Mastrapas, A. Koukounas, N. Wang, *et al.*, "jina-embeddings-v3: Multilingual embeddings with task lora," *arXiv preprint arXiv:2409.10173*, 2024.

[3] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, and C. D. Manning, "Raptor: Recursive abstractive processing for tree-organized retrieval," *arXiv preprint arXiv:2401.18059*, 2024.

[4] P. Islam, A. Kannappan, D. Kiela, R. Qian, N. Scherrer, and B. Vidgen, "Financebench: A new benchmark for financial question answering," 2023.

[5] Z. Jiang, X. Ma, and W. Chen, "Longrag: Enhancing retrieval-augmented generation with long-context llms," *arXiv preprint arXiv:2406.15319*, 2024.

[6] O. Press, N. A. Smith, and M. Lewis, "Train short, test long: Attention with linear biases enables input length extrapolation," *arXiv preprint arXiv:2108.12409*, 2021.
[7] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.
[8] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.
[9] J. Sun, S. Y. Min, Y. Chang, and Y. Bisk, "Tools fail: Detecting silent errors in faulty tools," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing* (Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, eds.), (Miami, Florida, USA), pp. 14272–14289, Association for Computational Linguistics, Nov. 2024.
[10] S. Hu, C. Lu, and J. Clune, "Automated design of agentic systems," 2024.
[11] D. Hendrycks, C. Burns, A. Chen, and S. Ball, "Cuad: An expert-annotated nlp dataset for legal contract review," *arXiv preprint arXiv:2103.06268*, 2021.
[12] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24824–24837, 2022.
[13] G. Kim, S. Kim, B. Jeon, J. Park, and J. Kang, "Tree of clarifications: Answering ambiguous questions with retrieval-augmented large language models," *arXiv preprint arXiv:2310.14696*, 2023.
[14] L. Luo, Y.-F. Li, G. Haffari, and S. Pan, "Reasoning on graphs: Faithful and interpretable large language model reasoning," in *International Conference on Learning Representations*, 2024.
[15] T. Rebedea, R. Dinu, M. Sreedhar, C. Parisien, and J. Cohen, "Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails," *arXiv preprint arXiv:2310.10501*, 2023.

APPENDIX

A. Datasets

The following section discusses the datasets used by us for the experimentation of the various strategies and approaches tested by us as discussed above.

1) Finance Bench: The Finance Bench dataset primarily includes questions that require financial reasoning, logical deduction, and quantitative analysis. These questions test models' ability to interpret complex financial statements, perform numerical calculations, and draw insights based on financial and market trends.

Example Queries

- **Question 1:** Assume that you are a public equities analyst. Answer the following question by primarily using information that is shown in the balance sheet: what is the year end FY2018 net PPNE for 3M? Answer in USD billions.
- **Question 2:** Does 3M have a reasonably healthy liquidity profile based on its quick ratio for Q2 of FY2023? If the quick ratio is not relevant to measure liquidity, please state that and explain why.
- **Question 3:** What is Amazon's year-over-year change in revenue from FY2016 to FY2017 (in units of percent and round to one decimal place)? Calculate what was asked by utilizing the line items clearly shown in the statement of income.

2) Finance 10K: The Finance 10K dataset uses mostly factual and information retrieval-based questions. It focuses on assessing models' ability to extract specific financial facts, recognize financial entities, and retrieve concrete information from financial documents. The dataset emphasizes factual knowledge about financial terms, regulations, and company-specific information.

Example Queries

- **Question 1:** What area did NVIDIA initially focus on before expanding to other computationally intensive fields?
- **Question 2:** What was the major factor contributing to the increase in sales, general, and administrative expense in fiscal year 2023?
- **Question 3:** How much is authorized for the repurchase of additional shares of common stock as of January 29, 2023?

3) Finance Multihop: The Finance Multihop dataset, manually created by us for multihop financial reasoning, challenges models to answer complex questions that require gathering and relating information from multiple contexts across different chunks. We needed this dataset to test the effectiveness of our retrieval pipeline. We utilized the 10K reports of the FinanceBench dataset to curate the questions of the Finance Multihop.

Example Queries

- **Question 1:** How did 3M's accounting treatment for its Venezuelan subsidiary's financial statements change in response to Venezuela's economic situation, and what factors did the company consider for using different exchange rates?
- **Question 2:** What factors contributed to MGM Resorts' consolidated net revenue increase and operating loss in the fourth quarter of 2022, and how did they differ across the Las Vegas Strip Resorts, Regional Operations, and MGM China?
- **Question 3:** How does the ongoing conflict in Ukraine impact PepsiCo's financial operations, and what measures does the company take to manage risks associated with international operations and commodity prices?

4) CUAD MultiHop: Much like the Finance Multihop dataset, we specifically curated this dataset for the legal domain using the openly available CUAD dataset. Some of the questions here are fact-based, requiring context from various pages, while others are more situation-based, requiring reasoning over the legal context. The need for this multihop dataset arose when we realized that the queries of the original CUAD datasets were not multihop and were unsuitable to test both the reasoning and retrieval aspects of our pipeline.

Example Queries

- **Question 1:** What defines 'Confidential Information' as per Section 10.1, and what are some exceptions where information is not considered confidential?
- **Question 2:** What were the key terms of the strategic alliance between Hyatt and Playa, and how does the right of first offer process work for development opportunities under this agreement?
- **Question 3:** What obligations does Transporter have regarding the design and modification of facilities, how are transportation and other service fees structured, and what is the payment process, including consequences for non-payment?

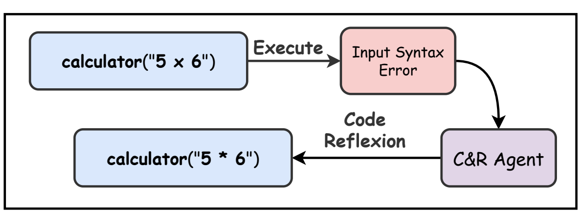
5) Open Australian Legal Q&A: This is an openly available legal dataset that was used to evaluate our pipeline on an openly available legal dataset.

Example Queries

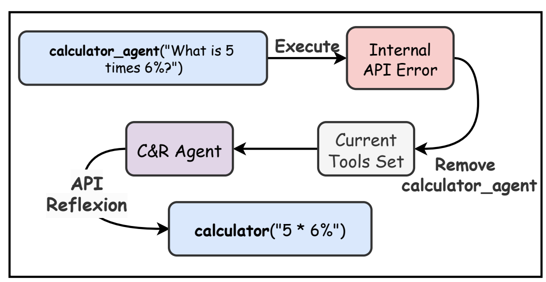
- **Question 1:** When did the Poisons List Amendment Order 2012 come into effect in Tasmania?
- **Question 2:** In the case of Angela Therese Harvey (nee Alecci) v Eileen Therese Alecci & Anor [2002] NSWSC 898, what was the court's decision regarding the order for mediation?
- **Question 3:** What are the three additional assets of BM that need to be included in the distributable pool according to the appellant's case in the decision of Iliopoulos v BM2008 Pty Ltd (In Liquidation) (ACN 005 762 685) [2010] FCA 787?

B. Tool failures & Reflexion

1) Failed Function Calls: The following illustrate the example of failed function calls due to (i) incorrect python syntax and (ii) internal API failure.



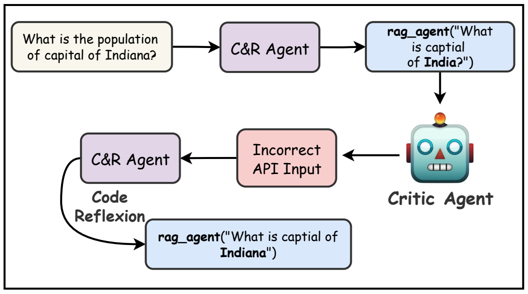
(a) Incorrect Python Syntax



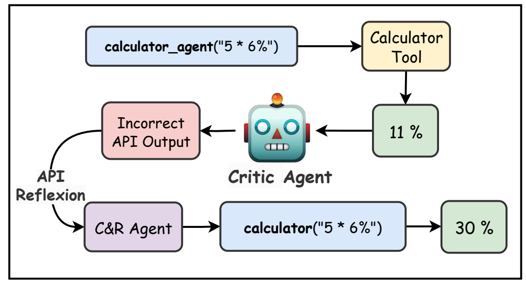
(b) Internal API Failure

Fig. 7: Illustration of unexecuted tool failure and reflexion

2) Executed Function Calls: The following illustrate the example of silent failure in executed function calls due to (i) incorrect input arguments and (ii) incorrect API output.



(a) Incorrect Input Arguments



(b) Incorrect API Output

Fig. 8: Illustration of executed tool failure and reflexion

C. Comparison Of RAG Agent Reasoning Technique

Following shows how the two reasoning approaches (Interleaving and Tree of Clarifications) tackle a question and try to arrive at the final answer.

Comparison of Approaches : Interleaving VS TOC

Question: What is Adobe's year-over-year change in unadjusted operating income from FY2015 to FY2016 (in units of percents and rounded to one decimal place)? Provide a solution by using the income statement.

INTERLEAVING APPROACH

Retrieval Thought: What is Adobe's unadjusted operating income for FY2015 and FY2016 according to the income statement?

Retrieval Observation: The unadjusted operating income for FY2015 and FY2016 is $903,095 and $1,493,602, respectively.

Reasoning: Let's calculate the year-over-year change in unadjusted operating income from FY2015 to FY2016.

1) First, calculate the difference in unadjusted operating income between FY2016 and FY2015:
$$1,493,602 - 903,095 = 590,507$$
2) Next, divide the difference by the FY2015 unadjusted operating income to get the change as a decimal:
$$\frac{590,507}{903,095} = 0.6538$$
3) Finally, multiply by 100 to convert to a percentage and round to one decimal place:
$$0.6538 \times 100 = 65.4\%$$

Final Answer: Adobe's year-over-year change in unadjusted operating income from FY2015 to FY2016 is 65.4%.

TREE OF CLARIFICATION

To calculate the year-over-year change in unadjusted operating income from FY2015 to FY2016, we need to first determine the unadjusted operating income for each year.

The unadjusted operating income for FY2016 is $2,721.8 (as per the detailed question-answer pair) and for FY2015 is $3,051.2 (also from the detailed question-answer pair).

Then, calculate the year-over-year change by dividing the difference in operating income ($2,721.8 - $3,051.2 = -$329.4) by the FY2015 operating income ($3,051.2), and then multiply by 100.

However, the correct year-over-year change is calculated as:

$$\left(\frac{2,721.8 - 3,051.2}{3,051.2}\right) \times 100 = -10.8\%$$

Since we are looking for a percent and the percent value calculated is a decrease (-10.8%), we can write the answer as 10.8 percent.

D. Comparison Of Tool Reasoning Technique

Following shows how the comparison of responses of C&R agent and ReAct over a tool-based query:

Comparison of Approaches : C&R Agent VS ReAct

Question: What is the average of NVIDIA's total comprehensive income in FY 2022 and FY 2023?
Tools: ragagent, calculator, ragagent1(Faulty), calculatoragent(Faulty)

C&R AGENT	REACT
	Thought: The current language of the user is: English. I need to use a tool to help me answer the question.
Function Call: ragagent("What was NVIDIA's total comprehensive income in FY 2022?")	**Action:** ragagent **Action Input:** 'query': "What was NVIDIA's total comprehensive income in FY 2022?"
Observation: NVIDIA's total comprehensive income in FY 2022 is $9,693.	**Observation:** Error
Function Call: ragagent("What was NVIDIA's total comprehensive income in FY 2023?")	**Thought:** he tool didn't provide the desired output. I need to try a different tool or approach. **Action:** calculatoragent **Action Input:** 'query': "Get NVIDIA's total comprehensive income in FY 2022 and FY 2023"
Observation: NVIDIA's total comprehensive income in FY 2023 was $4,377 million.	**Observation:** Error
Function Call: calculatoragent("What is the average of 9693 and 4377?")	**Thought:** The tool didn't provide the desired output. I need to try a different tool or approach.
Observation: API Failure	**Action:** ragagent1 **Action Input:** 'query': "What was NVIDIA's total comprehensive income in FY 2022 and FY 2023?"
Function Call: calculator("What is the average of 9693 and 4377?")	**Observation:** NVIDIA's total comprehensive income was $9,722 million in FY 2022 and $4,336 million in FY 2023.
Observation: 7035	**Thought:** I have the necessary information to answer the question. I'll use the user's language to answer.
Final Answer: The average of NVIDIA's total comprehensive income in FY 2022 and FY 2023 is $7,035 million.	**Answer:** The average of NVIDIA's total comprehensive income in FY 2022 and FY 2023 is ($9,722 million + $4,336 million) / 2 = $7,029 million.

E. Examples of Challenges Faced

1) Sparse Knowledge Graph Generation: The following shows an example of sparse KG generation from a large unstructured context (approx 1K tokens) through LLMs resulting in poor retrieval.

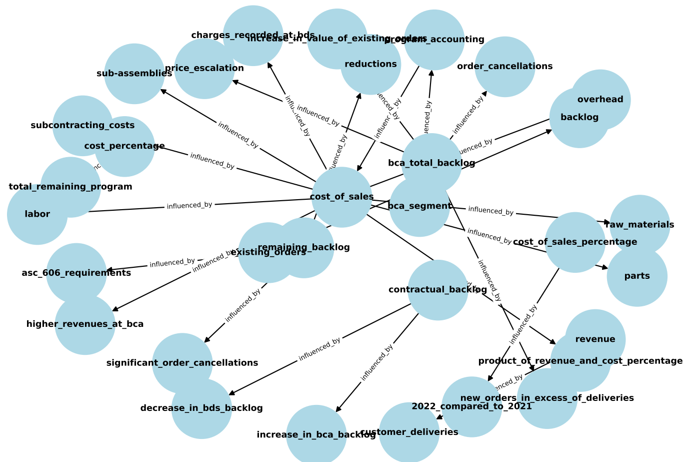


Fig. 9: Sparse KG generation using Llama 3.1 405B

2) Domain-specific Jargon Misinterpretation: The following example shows the failure of agent in dealing with jargons and domain specific terminologies, here *quick ratio* , when not provided in the context.

Jargon Misinterpretation

FLAWED REASONING

Question: Does 3M have a healthy liquidity profile based on its quick ratio for Q2 FY2023?

Retrieval Thought: What is the quick ratio of 3M for Q2 FY2023? The context does not provide the required data to calculate the quick ratio, which is determined by dividing cash, cash equivalents, and accounts receivable by total current liabilities.

Reasoning:

1) Quick Ratio = ($4,258 + $4,947) / $10,936 = 0.84

Final Answer: Yes, 3M has a healthy liquidity profile based on this quick ratio.

CORRECT REASONING

Question: Does 3M have a healthy liquidity profile based on its quick ratio for Q2 FY2023? Formula: Quick Ratio = (Total current assets - Total inventories) / Total current liabilities

Retrieval Thought: What are the values for current assets, inventories, and liabilities?

- Total current assets: $15,754M
- Inventories: $5,280M
- Current liabilities: $10,936M

Reasoning:

1) Quick Ratio = ($15,754 - $5,280) / $10,936 = 0.96

Final Answer: 3M has a healthy liquidity profile based on this quick ratio.