TCAF: a Multi-Agent Approach of Thought Chain for Retrieval Augmented Generation

Jun Zhao
junzhao@ebay.com
eBay
Shanghai, China

Xiaojiang Liu
xiaojiliu@ebay.com
eBay
Shanghai, China

Abstract

The increasing popularity of Retrieval Augmented Generation (RAG) with Large Language Models (LLMs) has highlighted the need for enhanced responses to user queries by leveraging web content knowledge. Despite its potential, the challenge of integrating noisy external web information often results in hallucination, and the issue of consistently providing correct answers remains unresolved. To improve this research, Meta introduced a comprehensive dataset CRAG and hosted the KDD Cup 2024 Challenge to advance RAG system development.

This paper details our solution in the competition, which consists of a three-component pipeline: Pre-processing, Retrieval, and Multi-Agent Generation. Our strategy incorporates Query Rewriting, Reference Constraint, and Conditional False-premise Detection to improve accuracy and reduce hallucinations. Moreover, we propose a novel "Thought-Chain-Agent-Flow" technique in the Multi-Agent Generation, enhancing the LLM's focus on critical facts and reasoning capabilities. This approach demonstrated superior performance, leading our team, bumblebee7[1], to won first place in the multi-hop challenge of Task1 and maintain top positions in Task2 and Task3, competing against over 2000 participants.

Keywords

Retrieval Augmented Generation, Large Language Model, Multi-Agents

1 Introduction

1.1 Background

Question Answering (QA) tasks[1, 4], such as those used in iPhone's Siri, are complex but vital for all information systems as they directly impact the user experience. These tasks demand precise comprehension of user questions and instant delivery of accurate answers. Recently, methodologies such as Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG)[5] have gained popularity in managing QA tasks, but they do come with issues like hallucination and inaccuracy. However, there are few benchmark and study exploring these issues.

[1]All authors contributed equally to this work.

To address this gap, Meta has released CRAG dataset[9] and organized the KDD Cup 2024 challenge with 3 different tasks, aiming to tackle the complex RAG problems with high-performance, low-cost solutions.

A team from eBay (the authors of this paper), participated in the competition and achieved a top position on the leaderboard, specifically winning the multi-hop question in Task 1. This paper outlines the team's solutions across all three tasks.

1.2 Dataset

For the challenge, Meta created the "CRAG: Comprehensive RAG Benchmark", a QA dataset consisting of 4,409 pairs of question and answer along with crawled web pages, all in English.

The dataset features eight question categories: simple, simple with condition, post-processing heavy, set, comparison, aggregation, multi-hop, and false premise. Table 1 describes the definition for each question category in detail.

Each crawled web page contains URL, web title, HTML content, page snippet (abstract), last update time, etc.

1.3 The Competition Tasks

The challenge had 3 tasks, hosted on AIcrowd:

Task 1: Retrieval Summarization This task involves predicting an answer given a question complemented by five web pages, each page have up to millions of chars. The goal is to provide an accurate response in short time while avoiding incorrect answers and hallucinations, which is the fundamental goal that 3 tasks and all typical RAG system shared.

Task 2: Knowledge Graph and Web Augmentation This task mirrors Task 1 but with the additional access to mock Knowledge Graphs (KGs) via mock APIs.

Task 3: Knowledge Graph and Web Augmentation This task mirrors Task 2 but with the additional feature with up to 50 pages. The goal is to further increase performance under excess data.

Our solution for the 3 tasks are very similar and can be described jointly, except for some extra recall process specific to Task 3 (Section 3.1).

2 Overall Framework

To facilitate the generation of accurate answers based on retrieved content, we have developed a structured framework for our Retriever-Augmented Generation (RAG) system, depicted in Figure. 1. This framework is designed to optimize the performance of each independent module, enhancing the overall effectiveness of the system. The framework comprises three main components: Pre-processing, Retrieval, and Multi-agent Generation.

Table 1: Definition of Question Categories

Question Type	Definition
Simple	Questions asking for simple facts that are unlikely to change overtime, such as the birth date of a person and the authors of a book.
Simple w. Condition	Questions asking for simple facts with some given conditions, such as stock prices on a certain date and a director's recent movies in a certain genre.
Set	Questions that expect a set of entities or objects as the answer (e.g., "what are the continents in the southern hemisphere?").
Comparison	Questions that compare two entities (e.g., "who started performing earlier, Adele or Ed Sheeran?").
Aggregation	Questions that require aggregation of retrieval results to answer (e.g., "how many Oscar awards did Meryl Streep win?").
Multi-hop	Questions that require chaining multiple pieces of information to compose the answer (e.g., "who acted in Ang Lee's latest movie?").
Post-processing heavy	Questions that need reasoning or processing of the retrieved information to obtain the answer (e.g., "how many days did Thurgood Marshall serve as a Supreme Court justice?").
False Premise	Questions that have a false preposition or assumption (e.g., "What's the name of Taylor Swift's rap album before she transitioned to pop?" (Taylor Swift has not yet released any rap album)).

(1) **Pre-processing**: This initial component involves several key processes including query rewriting, web page document segmentation, and rule-based filtering. The primary objective here is to refine the input raw data, ensuring it is clean and well-processed, which is useful for optimizing the accuracy of the subsequent modules.
(2) **Retrieval**: In this component, the system searches through a batch of web pages to identify potential answer candidates. It is imperative that this retrieval is comprehensive, capturing all necessary information so that the LLM can effectively deduce the correct answers from the relevant content.
(3) **Multi-agent Generation**: The final and most critical component of our framework involves a novel approach to answer generation, utilizing a multi-agent chain across several rounds of LLM generation. This method, named as Thought-Chain-Agent-Flow (TCAF), has proven particularly effective in addressing challenges such as hallucination, handling multi-hop questions, and producing coherent and contextually appropriate responses for users.

In order to enhance the readability and focus on the critical aspects of our implementation, we have structured the following system details into three key sections. We begin by discussing the "Retrieval" approach in Section 3. This is followed by an in-depth examination of our primary component, "Multi-Agents Generation" presented in Section 4. This section is pivotal as it elaborates on the novel approach and techniques employed in our research. Lastly, in Section 5, we delve into the "Pre-processing" for ensuring the reproducibility of our results.

3 Retrieval Approach

3.1 Recall

In our approach, we use an semantic-based recall that leverages pre-trained models. For each query and text chunk, we employ a text embedding model msmarco-MiniLM-L-12-v3 to compute the embedding and return the N-th nearest text chunk, determined by cosine distance in relation to the query embedding. In Task 3, an additional recall step is implemented to reduce latency, which is based on query and the web page titles to filter relevant web pages.

We have evaluated numerous embedding models from the MTEB leaderboard[2], which offers performance metrics including recall and ranking tasks, as well as filters to assess model parameter size. Our analysis reveals that, despite their larger size and greater embedding dimensions, models like Sentence-BERT [7] and GTE [6] do not significantly outperform their lightweight counterparts in terms of performance.

We've also tried keyword-based and sentence-level embedding recall. However, these modifications did not lead to performance enhancement. The complexity of the questions within the CRAG dataset could be a potential reason for this. Our observations suggest that compared to other methods, a semantic-based recall approach might be the most effective.

3.2 Ranking

A typical ranking module utilizes complex models such as deep neural networks along with query-doc cross-features, to select optimal candidates from the recall output, while current research[3] employs LLM models for ranking.

We've experimented with DeBERTa and LLM-Rank, but the improvements were marginal. One possible explanation for this limited enhancement is that we didn't fine-tune the rank separately, therefore, the impact of the open-source rank was limited.

4 Multi-Agents Generation

The Multi-Agents Generation is the core component of our system. This section is structured to methodically introduce the elements that constitute this system. Initially, in Section 4.1, we present the

[2]https://huggingface.co/spaces/mteb/leaderboard

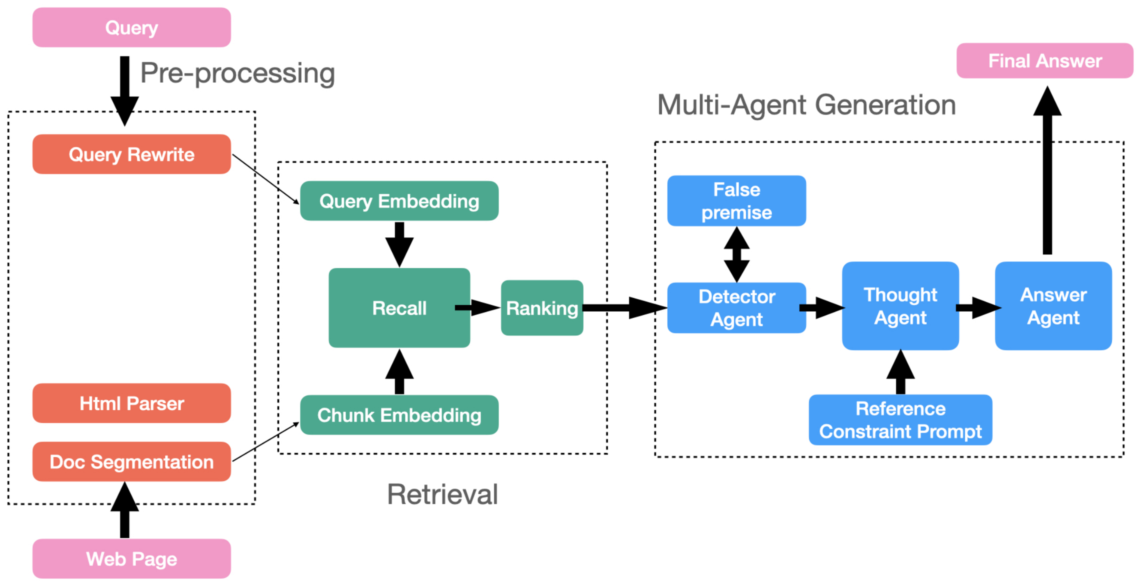


Figure 1: Our Retriever-Augmented Generation Framework in the KDD'25 Competition

novel algorithm "Thought-Chain-Agent-Flow", which is fundamental to our agent generation process. Subsequently, in Section 4.2, we explore "Reference Constraint", a strategic prompt engineering approach that enhances the accuracy. Finally we will illustrate the pre-train model in Section 4.3. Together, these sections provide a detailed insight into the Multi-Agents Generation, underscoring its importance in our system.

4.1 The Core: Thought-Chain-Agent-Flow

In our work, we introduce a novel method for answer generation based on a multi-agent flow approach. Our proposed method, referred to as the Thought-Chain-Agent-Flow, leverages the capabilities of multiple agents to handle complex deductive reasoning problems, particularly those involving multi-hop scenarios. Through comparative analysis with other agent-based approaches, the Thought-Chain-Agent-Flow demonstrated superior performance in the majority of test cases, showcasing its enhanced ability to manage intricate problem-solving QA tasks.

The implementation of the thought-chain-agent-flow is structured into three distinct stages:

(1) **Detector Agent**: This initial agent is designed to identify queries that may contain incorrect assumptions or premises, such as the query 'What is the price of Apple stock when they split the stock for the 10th time?' given that Apple has only split its stock five times. These erroneous queries can compromise the accuracy and reliability of the LLM's responses. To address this, we have deployed a dedicated detector agent tasked with identifying and validating any assumptions within a query.

(2) **Thought Agent**: This agent is responsible for processing and analyzing all retrieved content information. It synthesizes this data to draw conclusions and generates a set of detailed reasoning thoughts. These thoughts serve as an intermediate response, encapsulating the cognitive process that leads to a better answer.

(3) **Answer Agent**: Building upon the thought agent, the answer agent takes center stage in the third phase. It primarily focuses on the reasoning thoughts provided by the thought agent, using them as the main content for generating the final answer. The answer from the previous stage is also considered, but only as a weak reference, ensuring that the answer agent's output is predominantly influenced by the newly derived logical reasoning.

In the Answer Agent Stage, it is crucial to highlight that retrieved content information is intentionally omitted from subsequent readings by the LLM agent. This approach has been adopted to enhance the focus of the LLM on the most relevant information, thereby streamlining the context it needs to process. By reducing the cognitive load, this method not only augments the reasoning capabilities of the LLM but also minimizes the likelihood of forgetting issue during the generation process. For instance, the LLM may independently generate an accurate clarification fact, but could potentially overlook this detail in the presence of an extensive context. By maintaining a concise context, the Answer Agent effectively supports the LLM in consistently recalling and integrating key facts throughout the interaction, leading to more coherent and contextually appropriate responses.

It is also noteworthy that, our Thought-Chain-Agent-Flow (TCAF) approach, although conceptually similar to the Chain-of-Thoughts[8] method, exhibits significant differences in performance and stability. The Chain-of-Thoughts method operates on the principle of allowing the LLM to "think before reacting". However, this method often suffers from instability in the LLM's inference process, which can result in incomplete processing and, consequently, the absence of a final response. In contrast, our TCAF approach adheres more reliably to this principle, with each agent focusing on a specific sub-task. This structured focus enables the LLM to consistently produce high-quality and coherent responses.

Furthermore, empirical evidence from our experiments demonstrate that TCAF not only outperforms the Chain-of-Thoughts method in terms of stability but also provides a significant advancement in handling RAG tasks.

To ensure this, in our multi-agent framework, we implemented and evaluated several other agent roles to ensure a comprehensive and fair comparison of different methods. Despite integrating various roles into the framework, none matched the performance of the Thought-Chain-Agent-Flow (TCAF) approach. Consequently, these roles were not included in the final implementation. Notable roles tested but later excluded due to their comparative underperformance, which included:

(1) Summarization Role: An agent tasked with summarizing retrieved content without altering its meaning.
(2) Ranking Role: An agent responsible for ranking the most critical content to aid in answer generation.
(3) Relevance Role: An agent designed to assess the relevance of content to the posed question.

Among the tested solutions, the TCAF method consistently outperformed others. This led to the decision to exclusively incorporate the TCAF approach in our final system, eliminating other roles from the multi-agent framework.

4.2 Prompt Engineering: Reference Constraint

Within the framework of the TCAF, prompt engineering is another design that can significantly influences the RAG performance. Our investigation involved testing various prompt templates in competition tasks, leading to the identification of a particularly effective prompt, which we have termed the "Reference Constraint Prompt".

The Reference Constraint Prompt is specifically designed to instruct the LLM to base its responses on information explicitly drawn from the retrieved content. This approach significantly mitigates the risk of the model generating unfounded or "hallucinated" responses, thereby enhancing the reliability and accuracy of the output. The structure of this prompt is crafted to ensure that the LLM consistently seeks and references verifiable information from the provided content, aligning its responses closely with the available data. The prompt template is designed like the following in Figure. 2, which is divided into four distinct parts:

(1) Question Instruction: This segment involves a general explanation of the task. Our findings indicate that any prompt that effectively explains the task is suitable for this section.
(2) Input Context: This part includes segmented documents from the retrieved content, each labeled with a prefix [ID]. The IDs are incremental numbers ranging from 1 to K, where K represents the total number of retrieved documents. This labeling aids the LLM in identifying and referencing the most relevant document more efficiently.
(3) Reference Constraint: This critical component explicitly instructs the LLM to base its answers on referenced support from the retrieved content. By requiring the LLM to cite the ID of the supporting document in its response, this design significantly reduces the incidence of hallucination, ensuring that the answers are well-founded and verifiable.
(4) Output Formatting Instruction: Guidelines for how the response should be structured are provided in this section, ensuring consistency and clarity in the output format.

It is important to note that the Reference Constraint Prompt is specifically integrated into the thought agent within the TCAF framework. The answer agent operates independently of the original retrieval content, focusing solely on the processed inputs from the previous stage.

Implementing this prompt structure has demonstrated a significant improvement in the RAG system's performance on the CRAG dataset, with an absolute CRAG score increase of 3%+.

4.3 Pre-train Model

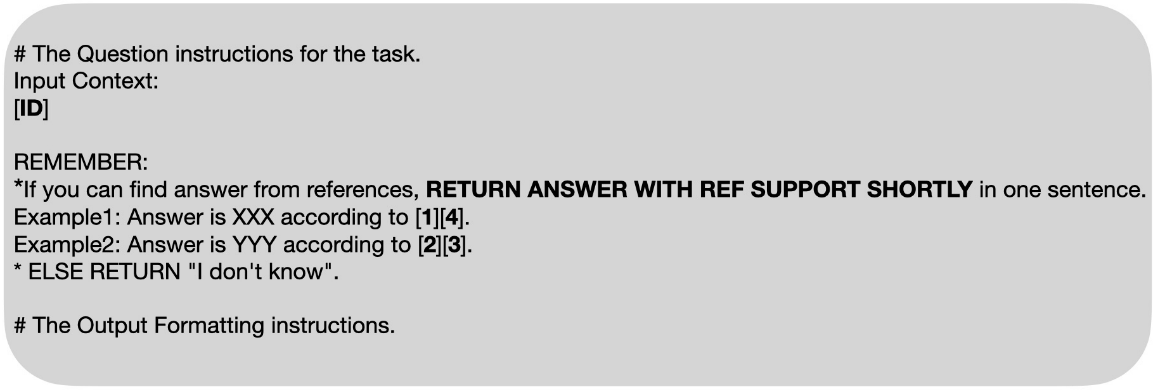
For our implementation, we consistently utilized the Llama-3-8B-Instruct model[2] across all multi-agent roles within the system. While alternative models, such as Llama2-70B and Llama-3-70B-Instruct, were also evaluated, the Llama-3-8B-Instruct model consistently demonstrated better performance for our TCAF method in the dataset, making it the preferred choice for our system. There are two possible reasons for this results:

(1) **Decomposed Tasks**: Our multi-agent TCAF method breaks down the RAG task into simpler sub-tasks, reducing the demand on the pre-trained LLM's ability. Consequently, it improves the performance of the Llama3-8B-Instruct model to match that of larger models, possibly leaving no room for further improvement merely by switching the pre-trained model.
(2) **Reduced Hallucination**: Although Llama-3-70B-Instruct is a more powerful model, it exhibited a tendency to generate hallucinations, especially when uncertain about responses. This issue was particularly prevalent with real-time changing questions in the dataset. Llama-3-70B-Instruct, having been exposed to a vast array of questions during training, often relied on its internal knowledge base, leading to answers that did not account for recent changes and thus were prone to inaccuracies.

Given these considerations, along with the distribution of question types and the overarching design of the multi-agent system, Llama-3-8B-Instruct was selected as the standard pre-trained model for all tasks and agents in our latest system version.

5 Pre-processing

Data pre-processing, primarily involving query rewriting and document processing, is noteworthy for a RAG system. These techniques aid the system in understanding queries and restructuring information in a more effective manner, consequently leading to enhanced system performance. We share the detail of pre-processing in this section for reproducibility.



The Question instructions for the task.
Input Context:
[**ID**]

REMEMBER:
*If you can find answer from references, **RETURN ANSWER WITH REF SUPPORT SHORTLY** in one sentence.
Example1: Answer is XXX according to [**1**][**4**].
Example2: Answer is YYY according to [**2**][**3**].
* ELSE RETURN "I don't know".

The Output Formatting instructions.

Figure 2: Reference Constraint Prompt Template

5.1 Query Rewrite

Query Rewriting is a commonly used practice in information retrieval systems to better understand a query's intent.

In our work, we used query rewriting for enriching a query with query_time information from our dataset. This tactic is designed to overcome the variability in question dynamics, which can span from days to years. For instance, a query like "what's the stock price of Apple last Monday" would be rewritten for time-specific accuracy, such as "what's the stock price of Apple on 20240805". However, a query like "who was the Oscar best actor winner in 2006?" would be left unchanged as it's already specific.

After a series of test, this technique has been shown to notable enhance the performance of RAG.

5.2 Document Pre-processing

To increase the performance of reference text chunk retrieval, for web pages provided by dataset, we implement document preprocessing. This step ensures the conversion of HTML pages into pure text chunks, discarding meaningless information such as HTML tags, CSS, and visual media elements like images. In our solution, a proficient HTML parser and effective chunk segmentation are details will be described in the following sections.

5.2.1 HTML Parser. We leveraged html2text[3] to parse HTML into ASCII text, configured to ignore hyperlinks, images, and decoding errors while including tables and external links. The output text formatting width is unlimited, and preference is given to inline links. Google-style presets are used for list indents and HTML anchors are ignored. In our configuration, html2text demonstrated greater efficiency compared to Beautiful Soup[4].

5.2.2 Chunk Segmentation. Once we have pure text from HTML pages, HTMLHeaderTextSplitter from langchain is used to split text into short chunks, which can be beneficial and increase performance in later operations such as recall and ranking. Two hyperparameters are defined and tuned: chunk_size (e.g. 200 1000 tokens) and chunk_overlap (e.g. 20 70 tokens), we find they would easily affect the final performance. Chunk_size is the size of each chunk of the text after splitting, while chunk_overlap controls the overlapping portions between consecutive chunks. This overlap concept is useful in scenarios where context from a section of text may be necessary to properly understand the subsequent text.

Following a comprehensive series of tests and empirical assessments, we opted for the parser, chunk_size and and chunk_overlap that showed exceptional performance. The optimal hyper parameter for the chunk_size is 700, while the chunk_overlap is 20.

6 Experiments

We calculate CRAG score through auto-evaluation process to measure performance of RAG system. For each question, with score 1, -1 for each correct and incorrect (hallucination) answer, where hallucinated answers are penalized. Then we calculate the average score of all samples as CRAG score. Additionally, we will consider accuracy and hallucination metrics as reference, which is the percentage of correct and incorrect answers.

6.1 Algorithm Comparison

In Tables 2 and Tables 3, we present a comparison of our method on Task 1 and Task 2 against ablation settings. All methods are based on RAG and the differences between each method are as follows:

Chain-of-Thoughts: This method serves as the baseline, utilizing the standard RAG approach by Chain-of-Thoughts prompt. It will be referred to as "CoT" hereafter.

CoT with pre-processing: This enhanced version of the baseline incorporates several key processes such as optimized query rewriting, web page document segmentation, and rule-based filtering. It will be denoted as "CoT(Prep)" in subsequent references.

CoT (pre-processing) + Reference Constraint: Building upon the pre-processing method, this approach integrates our Reference Constraint technique to further refine the process. It will be abbreviated as "CoT(Prep)+RC" in the following text.

TCAF + Reference Constraint (ours):: In this method, we replace the CoT method with our Thought-Chain-Agent-Flow (TCAF) approach while continuing to utilize Reference Constraint. It will be referred to as "TCAF+RC" henceforth.

When we compare CoT(Prep) with CoT, we can infer that preprocessing increases the accuracy (Task1 by +0.052, Task2 by +0.033)

[3]https://github.com/Alir3z4/html2text/tree/master
[4]https://beautiful-soup-4.readthedocs.io/en/latest

but also increase hallucination for Task1 (-0.04), and consequently slightly enhance the overall CRAG score (Task1 by +0.012, Task2 by +0.034).

When comparing CoT(Prep)+RC with CoT(Prep), we find that the Reference Constraint are effective in reducing hallucinations, as indicated by a decrease in Task1 (by -0.036) and Task2 (by -0.018). Also, there's an increase in accuracy for Task2 (by +0.015). Consequently, these combined effects result in an improved CRAG score (increase in Task1 by +0.036, and in Task2 by +0.033).

When comparing TCAF+RC with CoT(Prep)+RC, we observe a further increase in accuracy (Task1 by +0.009, Task2 by +0.015) as well as a reduction in hallucination (Task1 by -0.027). This successful combination significantly boosts the CRAG score (an increase of +0.036 in Task1 and +0.021 in Task2).

It is also worth mentioning that the Reference Constraint (RC) prompt technique can be readily applied to any RAG methods, including our TCAF, which consistently yields further improvements. In our experiments, we did not encounter any drawbacks when using RC, though further exploration of this aspect is left for future work. Consequently, the TCAF+RC method demonstrated the best performance in our experiments compared to all existing RAG methods.

These comparisons indicate that our solution performs the best in CRAG score while simultaneously tackling hallucination problems, demonstrating its superiority in complex QA tasks.

Table 2: Task 1 Performance

Approach	CRAG Score	Accuracy	Hall.
`Chain-of-Thoughts`	0.0236	0.134	0.110
`CoT (Prep)`	0.0360	0.186	0.150
`CoT (Prep) + RC`	0.0721	0.186	0.114
TCAF + RC	**0.1081**	**0.195**	**0.087**

Table 3: Task 2 Performance

Approach	CRAG Score	Accuracy	Hall.
`Chain-of-Thoughts`	0.0079	0.126	0.118
`CoT (Prep)`	0.0420	0.159	0.117
`CoT (Prep) + RC`	0.0751	0.174	0.099
TCAF + RC	**0.0961**	**0.189**	**0.093**

6.2 Pre-trained Model Analysis

Given that Task 1 and Task 2 yielded similar results, here we will use only Task 1 for further analysis.

We tested two different sizes of pre-trained models for our TCAF-RC approach and obtained surprising results. While larger LLMs typically improve outcomes, this trend does not always hold if a multi-agent method has already significantly enhanced QA capabilities. As indicated in Table 4, the Llamma3-70B version experienced a 0.02 drop in CRAG score. This demonstrates that well-designed roles in a multi-agent system can boost the performance of an 8B LLM to match that of a 70B LLM in the RAG system. The performance drop could be explained by Llamma3-70B's higher tendency to rely on its internal knowledge, leading to more hallucinations.

Table 4: Pre-trained model Test

Approach	CRAG Score
TCAF-RC(Llamma3-8B-Instruct)	0.1081
TCAF-RC(Llamma3-70B-Instruct)	0.0883

6.3 Leaderboard Rank

There are over 2000+ participants from various countries, and over 5500 submissions, we won the Top 8 in the leaderboard[5].

7 Final Winner Announcement

On 2024 July 27, the host announced the winners of the KDD Cup 2024 Meta CRAG Challenge. Different from evaluation method in Section 6, the evaluation process[6] combined assessments of GPT-4 auto-evaluation and human manual-evaluation, with weighting on different questions to reflect real-world QA use case, followed by a code validation.

We won the 1st of multi-hop challenge in Task1, with our score detailed in Tables 5. The multi-hop is questions that require chaining multiple pieces of information to compose the answer, for example: "Who is the American singer-songwriter who has won 13 Grammy awards and is known for her unique blend of pop, R&B, and electronic music, as well as her energetic live performances?" This question requires to find specific person chaining several clues: 1) American singer-songwriter 2) won 13 Grammy awards 3) known for certain performance and music style.

Table 5: Winner of Multi-hop in Task1

Rank	Question Type	Team	Score
1	multi_hop	bumblebee7	17.9

8 More Implementation Details for TCAF

In this section, we provide additional implementation details for the three-stage algorithms used in our multi-agent RAG system. The pseudocode for our TCAF-RC is illustrated in Algorithm 1.

[5]Refers to the leaderboard in Phase2b evaluated by GPT-4. The human judgement version for the leaderboard is not released yet.

[6]https://discourse.aicrowd.com/t/final-evaluation-process-team-scores/10785

Algorithm 1 TCAF-RAG

```
Input: a user query $\mathbf{q}$, a set of web pages denoted as $\mathbf{D_{page}}$, a query time $t$.
Output: The correct answer final_answer.
1:  Pre-process: Segment the set of web pages $\mathbf{D_{page}}$ into a set of chunks $\mathbf{D_{chunk}}$. Rewrite the query $\mathbf{q}$ to $\mathbf{q'}$, by taking into account the query time $t$.
2:
3:  Retrieval: Retrieve the top-k relevant chunks, denoted as $\mathbf{R_{chunk}}$, for the query $\mathbf{q}$ from the entire set of chunks $\mathbf{D_{chunk}}$.
4:
5:  Detector Agent: Try to extract the premise from the query $\mathbf{q}$.
6:  if The premise exists then
7:      if The premise is determined to be false based on $\mathbf{q}$ then
8:          return "Invalid Question".
9:      end if
10: end if
11:
12: Thought Agent: Incorporate $\mathbf{q'}$ and $\mathbf{R_{chunk}}$ into the Reference Constraint prompt template to create a complete prompt. Then, generate the thoughts and answer based on this consolidated prompt.
13:
14: Answer Agent: Generate the final_answer using a prompt that takes into account the thoughts and answer provided by the thought agent.
15:
16: return final_answer
```

It is important to note that for the majority of our implementation, we only utilized the zero-shot prompt template, which consistently delivers significant and robust results across all our experiments. In most instances, a zero-shot based prompt is sufficient for this work.

9 Conclusion

The Meta KDD Cup'24 competition presented a unique opportunity to tackle the challenges posed by the comprehensive CRAG dataset, which featured a diverse range of real-world queries and question types. Our approach involved constructing a robust RAG system through a systematic pipeline that included Pre-processing, Retrieval, and Multi-Agent Generation. Furthermore, our innovative "Thought-Chain-Agent-Flow" methodology significantly enhanced the accuracy and stability of our system. These strategies were instrumental in securing the 1st place in the multi-hop challenge of Task1 and achieving top positions on the Leaderboard of the Meta KDD Cup 2024 Challenge. Our success in this competition underscores the effectiveness of our methods and contributes valuable insights to the field of information retrieval and question answering systems.

References

[1] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to Answer Open-Domain Questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vancouver, Canada, 1870–1879. https://doi.org/10.18653/v1/P17-1171

[2] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783* (2024).

[3] Yupeng Hou, Junjie Zhang, Zihan Lin, Hongyu Lu, Ruobing Xie, Julian McAuley, and Wayne Xin Zhao. 2024. Large language models are zero-shot rankers for recommender systems. In *European Conference on Information Retrieval*. Springer, 364–381.

[4] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Matthew Kelcey, Jacob Devlin, Kenton Lee, Kristina N. Toutanova, Llion Jones, Ming-Wei Chang, Andrew Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural Questions: a Benchmark for Question Answering Research. *Transactions of the Association of Computational Linguistics* (2019). https://tomkwiat.users.x20web.corp.google.com/papers/natural-questions/main-1455-kwiatkowski.pdf

[5] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[6] Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards General Text Embeddings with Multi-stage Contrastive Learning. arXiv:2308.03281 [cs.CL] https://arxiv.org/abs/2308.03281

[7] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. http://arxiv.org/abs/1908.10084

[8] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[9] Xiao Yang, Kai Sun, Hao Xin, Yushi Sun, Nikita Bhalla, Xiangsen Chen, Sajal Choudhary, Rongze Daniel Gui, Ziran Will Jiang, Ziyu Jiang, et al. 2024. CRAG–Comprehensive RAG Benchmark. *arXiv preprint arXiv:2406.04744* (2024).