Final Report for CSE561 Fall 2024

Zheyuan Wu
Department of Mathematics
Washington University in St.Louis
1 Brookings Dr, St. Louis, MO 63130
w.zheyuan@wustl.edu

Abstract

We introduce a novel framework called Flow of Thoughts that allows a large language model (LLM) to retrieve information from self generated knowledge graph to solve problems without relying on additional pre-training. The framework is designed to dynamically iterate over a large data corpus, intelligently filtering and aggregating relevant information to form a comprehensive solution. Website and code are available at https://github.com/Trance-0/Flow-of-Thoughts.

1 Introduction

This final project focuses on studying how to help LLMs store and manage "knowledge" and "references". The large language model is a natural compressor [3]. We can efficiently compress the knowledge as a graph node with a set of tag tuples for further referencing. We can scale the data and keep the relevant information to solve the problem when prompting the LLM.

The memory problem of LLM has persisted for a long time. Some recent research finds that modifying where important information is placed within the language model's input context—such as moving the section that answers a question—creates a U-shaped performance pattern. The model performs better when this information is at the very start (primacy effect) or at the end (recency effect) of the input context, but its performance drops notably when the information is situated in the middle of the context. [6]

However, in real life, we need to study and gather tons of information when solving problems, an agent must be aware of many aspects of a question before making correct and consistent decisions. Increasing the memory size or other methods that help LLMs to gain information in the large corpus is essential to make the models solve problems like a human expert.

2 Related work

2.1 Chain-of-thoughts

Chain-of-thoughts [8] is an effective prompting method discovered in 2023 by Jason Wei etc. It provides an example of a logic chain to solve a problem similar to the target question allowing the LLMs to think step by step. This prompt can be used in the Training and Prompting stage of the LLM and generally provides better results when dealing with problems in Mathematics and Engineering.

One significant constraint faced by LLMs based on CoT is the context window size. This can lead to situations where the model forgets previous steps when working through complex problems, particularly in scenarios that require Chain-of-Thought (CoT) reasoning.

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

2.2 Graph-of-thoughts

Graph of thoughts [2] is a framework used to prompt LLMs by collecting the thinking process and branching different ideas generated by LLMs. Using generation and aggregation, the model selects the best result in the thinking process and develops on that.

integrating concepts such as the Graph of Thoughts and other search methods might provide valuable support for solving problems with a limited context window. By employing these techniques, we can enhance the model's ability to organize and retrieve relevant data, facilitating better problem-solving even with constrained memory resources.

However, during the aggregation process, the LLM cannot freely choose the information they need to solve the problem but just propagate from previous thoughts, in this research, our framework will try to give the LLM to choose the material that they find helpful.

2.3 Express uncertainty

Another relevant approach is detailed in a paper discussing the certainty of LLMs in problem-solving [5]. This research focuses on how models express and handle uncertainty, which can be instrumental in determining when to terminate the search or prompting process. Specifically, the authors explore techniques such as uncertainty sampling and confidence thresholds, which allow models to quantify their level of certainty about generated outputs. Understanding and incorporating measures of certainty can help optimize when and how the model utilizes its context, thereby allowing it to defer to more reliable responses or request additional information when faced with ambiguous queries. Understanding and incorporating measures of certainty can help optimize when and how the model utilizes its context, potentially leading to more accurate and efficient problem-solving.

2.4 Self-verification

Self-verification [9] is an important technique in improving the reliability of responses generated by large language models (LLMs). When an LLM initially produces an answer, there may be some inaccuracies due to the model's limitations in understanding the full context or providing detailed reasoning. However, the model has the ability to correct or refine its output by "thinking twice." One way this can be achieved is by prompting the model to reverse the roles of questions and answers. By taking the original answer and transforming it back into a question, followed by asking the LLM to generate an updated or revised answer, researchers can often obtain a more accurate and thoughtful response.

This process encourages the model to evaluate its earlier reasoning and detect inconsistencies or gaps in logic that may have been overlooked initially. The technique leverages the model's own knowledge to reassess and validate its responses, thereby functioning as a form of internal feedback. Additionally, this self-verification approach may prompt the model to consider alternative interpretations of the question, helping to mitigate issues like oversimplification or misunderstanding of complex queries. By iterating in this manner, the quality of the response can be significantly enhanced, offering a more reliable and nuanced answer for the user.

2.5 Memorizing Transformers and Self-Reflective Retrieval-Augmented Generation (SELF-RAG)

Furthermore, exploring memorizing transformers and their approaches could provide additional strategies for extending Transformer architectures using kNN [10]. These methods focus on enhancing the model's ability to retain and recall information across longer contexts, potentially offering practical solutions for memory limitations.

Other frameworks like Self-RAG [1] are also helpful for LLM to retrieve essential information when solving problems in long paragraphs. The model incorporates a feedback loop where it reflects on its own generated responses to improve their quality before delivering them. This reflection can involve: Checking for consistency with the retrieved documents, verifying the factual accuracy, and identifying potential hallucinations (when the model generates incorrect or fabricated information).

3 Framework design

We want to create a Flow of Thoughts framework that can dynamically iterate the long passage with self-RAG based on a graph of thoughts. The idea goes as follows:

First, the LLM will split the passage into several syntactically independent paragraphs by iterating the long texts. For example, the LLM will split a paper into sections like abstract, importance of projects, related work A, related work B, proposed solutions, etc that can fit into the context windows.

Then we let LLM to determine if the passage is relevant to the problem that we are going to solve. For example, when we ask "What related technologies did the author use when doing the project?" The LLM should ignore the proposed framework, conclusion, and experimental results section and only focus on reading the "Related work" sections.

Finally, we let LLM compose answers based on the related paper segments with supporting references.

Algorithm 1 Flow of thoughts(P, Q)

Require: Generator LM $\mathcal{M}$
Require: Large-scale passage collections $P = \{d_1, ..., d_N\}$
Require: Final question Q.
  $methods \leftarrow LM$(How to solve the problem Q, number of methods) ▷ Generate methods to solve the problem
  $solutions \leftarrow []$
  for each $method \in methods$ **do**
    $segments \leftarrow []$
    $mistakes \leftarrow LM$(common mistakes in $method$) ▷ Extract segments from the knowledge graph to obtain relevant information
    while P is not empty **do**
      $current\_passage \leftarrow P.pop()$
      $current\_segement \leftarrow \mathcal{M}($ relevant info in $current\_passage$ to solve problem $Q))$
      if current_segment is not empty **then**
        $segments$.add($current\_segment$)
      end if
    end while
    $thoughts \leftarrow []$ ▷ Aggregate the segments to form a rudimental solution
    for each $current\_segment \in segments$ **do**
      $thoughts$.add($current\_segment$)
    end for
    while $thoughts.size() > 1$ **do**
      $thought_a, thought_b \leftarrow$ first two solution of $thoughts$
      $current\_thought \leftarrow \mathcal{M}($ aggregate $thought_a, thought_b)$
      $refined\_thought, mistakes \leftarrow LM$(refine $current\_thought, mistakes$)
      $thoughts$.add($refined\_thought$)
      $mistakes$.add($mistakes$)
    end while
    $solutions$.add($thoughts$)
  end for
  return $majority(solutions)$

The framework is designed to be a general framework that can be applied to any problem that can be solved by LLMs.

3.1 Novelty of the solution

The Flow of Thoughts framework introduces a new approach to processing and extracting relevant information from a large data corpus that might be space for the necessary information to solve the target problem using Self-RAG and a Graph of Thoughts structure, significantly enhancing the efficiency, relevance, and quality of interactions with long-form content.

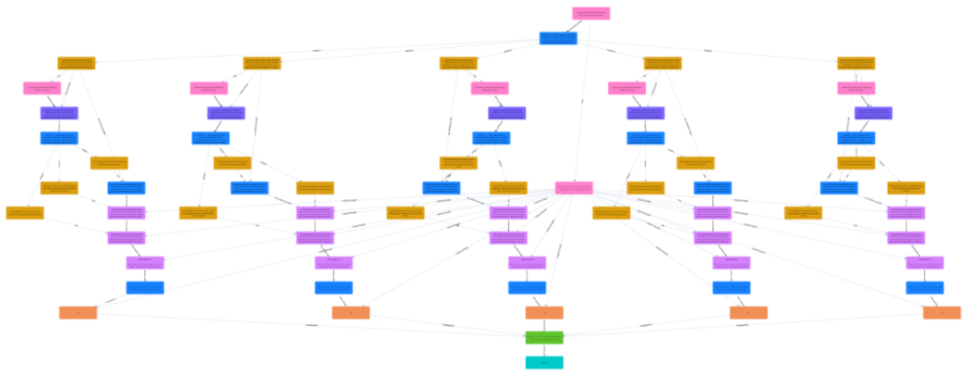

Figure 1: Flow of Thoughts framework using graph visualization, the pink nodes are the nodes from the knowledge graph or definition of the problem, the yellow nodes are generated by the LLM, the purple nodes are refinement of the solution generated by the LLM. In the final layer, the LLM will aggregate the solution and generate the final answer (the cyan node)

This model provides more explainability by recording how LLMs get the final solution from aggregating the partial information gained from Self-RAG. Through its intelligent segmentation, contextual relevance filtering, and ability to compose well-supported answers, this framework can potentially stand out in the landscape of text processing and information retrieval technologies.

3.2 Efficiency

Compared with a normal Graph of Thoughts, the framework proposed independent approaches to solve the problem and solve it automatically with self checking for common mistakes and refine the solution along each step. This use generation of LLM more effectively since each path is independent and the LLM can generate more diverse solutions compared to the normal Graph of Thoughts or Tree of Thoughts.

Moreover, the framework intelligently filters irrelevant sections based on the posed question and methods, enabling the model to focus solely on pertinent information. This specificity enhances the accuracy of the responses generated by the LLM and saves costs when dealing with large data corpus.

Compared with a normal Self-RAG Inference, the framework provides more flexibility for the convergence of information for black-boxed models like ChatGPT and Claude. It's easier to migrate to a new model without training the retriever and fine-tuning costs.

4 Experiment results

Now we have implemented basic functions for the Flow of Thoughts in revised Graph of Thoughts framework and let the LLM decide whether to use those messages or not when aggregating the final answer.

4.1 Sorting

The sorting task is a basic task that can be solved by Graph of Thoughts. we give the LLM a list of random numbers and let it sort them.

4.2 Set Intersection

The set intersection task is a basic task that also can be solved by Graph of Thoughts. We have implemented the basic functions for the Flow of Thoughts with the same functionality as Graph of Thoughts and compare the performance with Graph of Thoughts.

We use the same dataset as the Graph of Thoughts and compare the performance with Graph of Thoughts. However, due to time constraints, we only have the set intersection task results for length 64 and 100 trials using ChatGPT-4o and Claude-3.5-sonnet models.

4.3 Reading Comprehension

For the reading comprehension task, we use the RACE dataset [4] to test the performance of the Flow of Thoughts framework. The dataset contains 28,000+ passages and 100,000+ questions, which is a large dataset for reading comprehension tasks. We use the top 100 passages with longest length to test the performance of the Flow of Thoughts framework. (min passage length 3850, detail of the passages can be found in the appendix)

5 Analysis

6 Limitations and potential future work

6.1 Approach generation

The generated methods may still not be approachable for LLM to execute (Shor's algorithm to factorize large primes)

Among all the generated methods, most of them are approachable for LLM to execute and will not affect the major voting procedure of the final answer. However, this statement may not hold for more complex problems. The model may need access to external knowledge or API to execute some of the methods. One further improvement is to let the LLM access to external knowledge or API to execute some of the methods as described in ToolLLM [7].

6.2 High expense for graph generation

During the experiment, we found that the generation of the graph is expensive. The generation of the graph is $O(nm)$ where n is the number of approaches and m is the number of checking steps for each method. (Maybe we can share and check some common mistakes to save costs for improvements) We tried to reuse some of the common mistakes to save costs for improvements. However, the performance is still non-linear especially when the number of mistakes is large.

References

[1] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-rag: Learning to retrieve, generate, and critique through self-reflection, 2023.

[2] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690, March 2024.

[3] Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. Language modeling is compression, 2024.

[4] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017.

[5] Stephanie Lin, Jacob Hilton, and Owain Evans. Teaching models to express their uncertainty in words, 2022.

[6] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts, 2023.

[7] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023.

[8] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Huai hsin Chi, F. Xia, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *ArXiv*, abs/2201.11903, 2022.

[9] Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification, 2023.

[10] Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers, 2022.

7 Appendix

7.1 Prompts used in solving the problem

7.1.1 Sorting

For approach generation, we use the following prompt:

Example output:

7.1.2 Set Intersection

7.1.3 Reading Comprehension

Passages used for reading comprehension test can be found in github repository `https://github.com/Trance-0/Flow-of-Thoughts/tree/main/flow-of-thoughts/test/reading_comprehension/RACE_min`

For approach generation, we use the following prompt:

Example output: