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.

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 propose a novel Flow of Thoughts framework that can dynamically process and reason about long passages using self-RAG based generation combined with knowledge graph construction. This framework enables large language models to effectively manage and utilize information from extensive texts while maintaining coherent logic chains. The framework consists of four main components:

3.1 Preprocessing

First, the LLM performs intelligent text segmentation by splitting long passages into syntactically and semantically independent units that fit within the model's context window.

For academic papers, this involves breaking down the text into logical sections such as abstract, introduction, related work, methodology, and results. Each section is further subdivided if needed to ensure it fits within context limits while preserving meaningful relationships. For reading comprehension tasks, the framework segments passages into coherent paragraphs that maintain narrative flow and topical consistency while staying within context window constraints.

3.2 Generating methods

The framework then employs a multi-strategy approach by having the LLM generate diverse solution methods tailored to the specific problem type. For example, when tackling sorting problems, the LLM generates multiple sorting algorithms like bubble sort, quick sort, and merge sort.

For each method, the LLM also performs an error analysis by identifying potential pitfalls and edge cases that could arise during execution for future refinement.

3.3 Self-verification and refining

On each branch of the solution, the framework feeds the problem statement, generated methods, and data corpus to the LLM. The model evaluates passage relevance and use the generated methods to solve the problem. The LLM will also check if the generated thought follows the generated methods and the problem statement using self-verification [9] and iteratively check the potential mistakes generated in the previous steps.

If new errors are found, we may also add the new errors to the checking process.

For instance, when answering questions about technical approaches used in a project, the LLM selectively focuses on relevant sections like "Related Work" while appropriately discounting less relevant sections like conclusions or experimental results. This targeted attention mechanism helps maintain solution quality while managing computational resources effectively.

Ideally, in this step, the LLM should take control for the thought generation and refinement process and should be able to determine the steps required to solve the problem. However, in this research, we don't have sufficient time to implement such protocol, which can be done in the future. 6.4

3.4 Aggregation

The final phase aggregates the final thoughts from each independent branch to reach the final answer. Unlike Graph of Thoughts approaches that merge partial solutions, our framework uses a majority voting system to determine the final answer.

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

3.5 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.

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 $majority(solutions)$
```

This model provides more explainability by recording how LLMs get the final solution from aggregating the partial information gained from Self-RAG. Through the 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.

3.6 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

To test the performance of the Flow of Thoughts framework, we have implemented basic functions for the Flow of Thoughts in revised Graph of Thoughts framework with implementation of basic

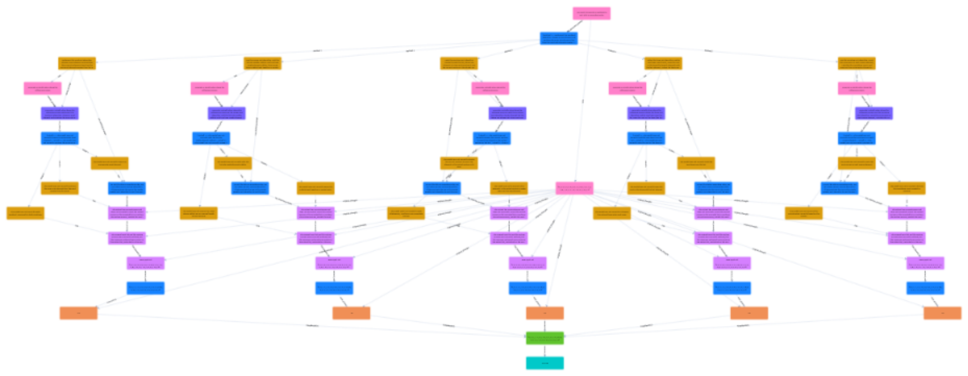


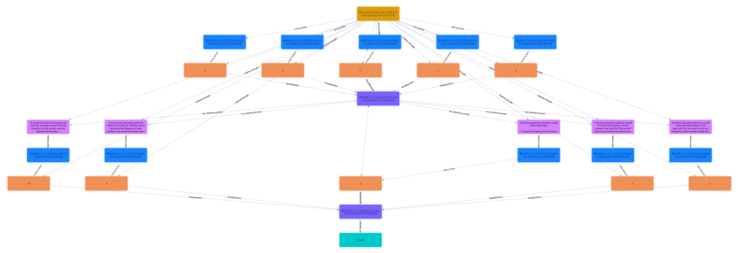
Figure 1: Flow of Thoughts framework using graph visualization, the pink nodes are the nodes from the knowledge graph or definition of the problem, the yellow nodes are generated by the LLM, the purple nodes are refinement of the solution generated by the LLM. In the final layer, the LLM will aggregate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

input and output (IO), Chain-of-Thought (CoT), Tree-of-Thought (ToT), Graph-of-Thought (GoT), and Flow-of-Thought (FoT). All the prompts are given with 3-shot examples. For the flow of thoughts framework, the shots are given three simple tasks with potential approaches.

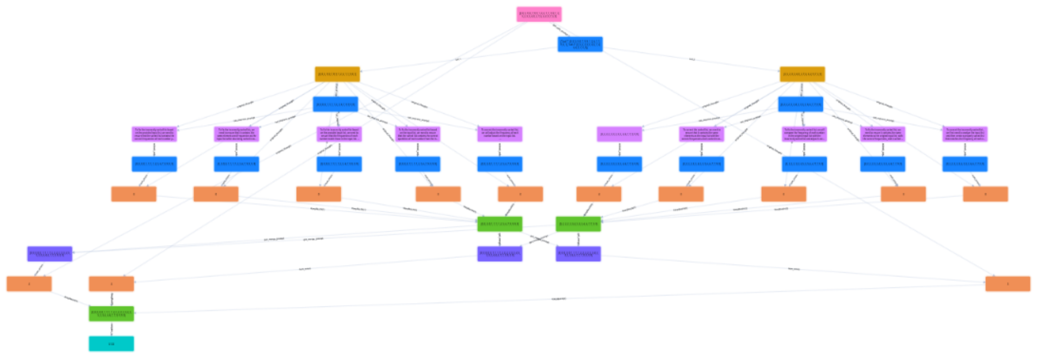
We use black-box models GPT-4o-mini and Claude-3.5-Sonet with API calls for the experiment. For both models, we set temperature to 0.7 with all other parameters to default.

(a) IO

(b) CoT



(c) Tree-of-Thought scheme



(d) Graph-of-Thought scheme

Figure 2: Other framework structure defined in the experiment, only use for reference, detailed structure can be found in each run of the experiment

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 32 random numbers and let each scheme to sort the numbers. The number of trials is 100. The number of errors is calculated by the number of trials that the LLM cannot sort the numbers correctly. Dataset can be found in appendix 7.1.1.

All the prompts are given with 3-shot examples. For the flow of thoughts framework, the shots are given three simple tasks with potential approaches.

Model	GPT-4o-mini	Claude-3.5-Sonet
IO	2.14	2.03
CoT	1.88	1.86
ToT	1.20	1.32
GoT	1.10	1.32
FoT	1.13	**1.09**

Table 1: Average number of incorrect responses in sorting 32 intergers

4.2 Set Intersection

The set intersection task is a basic task that also can be solved by Graph of Thoughts. The task is defined as follows: Given two set of 32 numbers, find the intersection of the two sets.

We use the same dataset as the Graph of Thoughts and compare the performance with Graph of Thoughts. Dataset can be found in appendix 7.1.2.

The experiment is conducted with 100 trials. The number of errors is calculated by the the number of misclassified elements in the intersection.

Model	GPT-4o-mini	Claude-3.5-Sonet
IO	3.02	2.88
CoT	2.46	1.89
ToT	1.93	1.97
GoT	1.24	1.56
FoT	1.13	1.20

Table 2: Average number of incorrect responses in set intersection

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)

The experiment is conducted with 100 passages. Each passage weights 1 point and we calculate the total score of the model by summing up the scores of each passage.

5 Analysis

5.1 Method generation

By default, we let the LLM generate 3 methods for each task all by themselves. However, the model may not be able to generate 3 practical methods that LLM can execute.

Model	GPT-4o-mini	Claude-3.5-Sonet
IO	68.5	65.25
CoT	72.33	69.5
ToT	76.25	75.33
GoT	78.5	83.25
FoT	84.33	85.5

Table 3: Score of the reading comprehension task on RACE-min dataset

For sorting task, in some rare cases, the LLM will generate methods using python code. However, in our setting, the LLM do not have the capability to execute python code and the methods will be processed as a normal tree of thoughts branch and don't contribute to the diversity of the solutions. This case is also observed in the set intersection task, where the LLM will also use `set.intersection()` to solve the problem.

For reading comprehension task, instead of generating 3 methods, the LLM will sometimes generate 1 or 2 methods only or even generate methods that are not related to the task. For instance, the LLM will produce Chunking and Visualizing as methods for the reading comprehension task, where you break down large pieces of information into smaller, more manageable sections (chunks). However, the LLM do not have the capability to generate image or diagram for the passages that can be feed to itself.

5.2 Self-checking process

The self-checking process is a crucial part of the Flow of Thoughts framework. It allows the LLM to check the correctness of the methods and refine the solution. However, the self-checking process is not perfect and the LLM may not be able to check the correctness of the methods.

For sorting task, we observe that the LLM cannot count the number of elements in the list correctly when the number of elements is large. The LLM will failed to count the number of missing elements in the list and snowball the error. For example, the LLM will generate 31 as the number of missing elements in the list when the correct number elements is 32. The LLM sometimes failed to check the length of the output list and use the illusions to fill the gaps of the list.

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 thoughts 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.

6.3 Flexibility for Divide and Conquer approach

The final Flow of Thoughts framework should allows the LLM to generate additional methods to divide the problem into sub-problems. However, the current framework is not flexible for this feature due to time constraints for this project. During the sorting process in Flow of Thoughts scheme, the LLM cannot divide the problem into sub-problems as normal Graph of Thoughts do, which might potentially hinder the performance of the framework for longer array size.

In future work, it's possible to design a protocol to let the LLM decide whether to divide the problem into sub-problems or generate additional filtering functions to filter out the invalid or incorrect solutions.

Our current Flow of Thoughts framework also don't fully support the self-retrieval process. The LLM will not be able to retrieve the information from the previous steps if those information is not stored in previous steps. Additional protocol is needed to enable the self-retrieval process as LLM needs.

6.4 Potential improvements that can be made

Due to time constraints, we only implemented the basic version of the Flow of Thoughts framework. There are many potential improvements that might be done to improve the performance of the framework.

Instead of finite steps generations, we can let the LLM continue to generate thoughts until we find a solution that LLM agree with with high confidence by adding certainty attribute [5] to each thought and use priority queue to check the most promising thoughts first.

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

Dataset for sorting can be found in `https://github.com/Trance-0/Flow-of-Thoughts/blob/main/flow-of-thoughts/test/sorting_test/sorting_032.csv`

Prompt for sorting can be found in `https://github.com/Trance-0/Flow-of-Thoughts/blob/main/flow-of-thoughts/test/sorting_test/sort_prompt.json`

7.1.2 Set Intersection

Dataset for set intersection can be found in `https://github.com/Trance-0/Flow-of-Thoughts/blob/main/flow-of-thoughts/test/set_intersections/set_intersection_032.csv`

Prompt for set intersection can be found in `https://github.com/Trance-0/Flow-of-Thoughts/blob/main/flow-of-thoughts/test/set_intersections/set_intersection_prompt.json`

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`

Prompt for approach generation can be found in `https://github.com/Trance-0/Flow-of-Thoughts/blob/main/flow-of-thoughts/test/reading_comprehension/reading_comprehension_prompt.json`