Mid-Term 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

Project choice: Designing a novel algorithm to do inference on large language models (white box models such as LLaMA2 models, or black box models such as GPT-4, CLAUDE, etc.) to solve some type of complex problems, and analyze their limitations.

1 Main goal of project

To investigate how to allow a large language model (LLM) such as GPT-4o to retrieve information relevant to solving problems without relying on additional pre-training. However, there is no free lunch for any improvements we make, we must try to utilize the context window as the bandwidth to transfer information as much as possible to utilize the intelligence of LLM to its maximum.

This final project focuses on studying how to help LLMs store and manage "knowledge" and "references". The large language model is a natural compressor [**?**]. 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.

2 Importance of project

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. [**?**]

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.

3 Related work

3.1 Chain-of-thoughts

Chain-of-thoughts [**?**] 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.

3.2 Graph-of-thoughts

Graph of thoughts [**?**] 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.

3.3 Express uncertainty

Another relevant approach is detailed in a paper discussing the certainty of LLMs in problem-solving [**?**]. 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.

3.4 Self-verification

Self-verification [**?**] 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.

3.5 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 [**?**]. 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 [**?**] 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).

4 Proposed Framework

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 $Q$.
  $segments \leftarrow []$
  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
  $solutions \leftarrow []$
  for each $current\_segment \in segments$ do
      $solutions$.add($current\_segment$)
  end for
  while $solutions.size() > 1$ do
      $solution_a, solution_b \leftarrow$ first two solution of $solutions$
      $current\_solution \leftarrow \mathcal{M}($ aggregate $solution_a, solution_b)$
      $solutions$.add($current\_solution$)
  end while
  return $solutions[0]$
```

4.1 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. 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 stands out in the landscape of text processing and information retrieval technologies.

4.2 Efficiency

Compared with a normal Graph of Thoughts, the framework intelligently filters irrelevant sections based on the posed question, 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.

5 Current experiment results (both positive and negative ones are acceptable) and insights gained from the results

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

We have tested that works for the same type of problem that Graph of Thoughts can solve with approximately the same accuracy in the keyword counting task. In the next step, we are trying to adapt the more complex task to fully utilize the advantage of the Flow of Thoughts framework, for example, the reading tasks used in Self-RAG to compare normal Self-RAG and our algorithm. For example, we might use Closed-set tasks[**?**] which include two datasets about public health and a multiple-choice reasoning dataset created from scientific exams to test the effectiveness of aggregation in our framework.

We are also planning to add certainty states [**?**] that help the framework to track the fuzziness of our created thoughts and double check on them when necessary,

During the practical coding process, we faced some difficulties in adapting our framework to the Graph-of-Thought framework since it was designed in a pipeline pattern that doesn't support the model to backtrack the information from previous thoughts, which might hinder the flexibility of our models. In the next month, we might try to free the communication between thoughts generated by our framework and proceed with more diverse connections between them.