An Iterative Prompting Approach to Graph-Theoretic Reasoning for Large Language Models

A minor thesis submitted in partial fulfillment of the requirements for the degree of Masters of Applied Science (Information Technology)

Name: Shengqing Jin
School of Computer Science and Information Technology
Science, Engineering, and Technology Portfolio,
Royal Melbourne Institute of Technology
Melbourne, Victoria, Australia

November 3, 2024

Declaration

I certify that, except where due acknowledgment has been provided, this work is solely my own and has not been submitted previously, in whole or in part, for any other academic award. The content of this thesis reflects work conducted since the official commencement date of the approved research program. Any editorial contributions by third parties, whether paid or unpaid, have been acknowledged, and all ethical procedures and guidelines have been followed throughout this research.

This work has been carried out since February 2024, under the supervision of Supervisor: Estrid He.

Name: Shengqing Jin
School of Computer Science and Information Technology
Royal Melbourne Institute of Technology
November 3, 2024

Acknowledgments

I extend my sincere thanks to RMIT and my supervisor, Dr. Estrid He, for her invaluable guidance, directional support, and willingness to answer countless questions along the way. Her insights and encouragement have greatly shaped this research. I am also grateful to Patrick Taylor for his assistance in setting up API configurations and navigating the constantly evolving Azure API environment, which presented a number of technical challenges. Their support has been instrumental to the successful completion of this work. My sincere gratitude extends to all those who supported this journey, making this challenging yet fulfilling process possible.

Special thanks to my wife Helen Zhu, Her unwavering support and encouragement have carried me through this challenging time, standing by me through this challenging, unpredictable life.

Contents

Abstract

This research investigates the enhancement of problem-solving capabilities in large language models (LLMs) for graph theory tasks through a CR prompting strategy. This study introduces CR prompting to improve the stepwise reasoning capabilities of LLMs. The CR technique facilitates a structured approach to question breakdown, where each reasoning layer incrementally builds on prior insights. This approach yielded a notable 7-12% improvement in problem-solving accuracy over Chain-of-Thought (CoT) prompting methods, achieved without altering model architecture or parameters, solely through refined prompting strategies. By demonstrating the potential of CR in enhancing LLM reasoning performance in specialized domains, underscoring its implications for further applications in graph theory and beyond.

Chapter 1

Introduction

1.1 Motivation of Research

LLMs have recently demonstrated great potential across various structured tasks, yet their effectiveness in handling complex, graph-based reasoning remains limited. Graph reasoning tasks are inherently multi-layered, requiring models not only to understand relationships but also to engage in iterative, cumulative thinking to navigate dependencies within the data. Traditional prompting techniques, such as CoT, have improved LLMs' reasoning abilities by introducing step-by-step processing of linear or hierarchical data. However, these techniques lack the depth needed to consistently manage the recursive, interdependent logic required in graph-based problems, where single-step approaches are insufficient.

Given the limitations of standard techniques, I believe there is a compelling need to explore alternative prompting methods that can guide LLMs through iterative cycles of reasoning, accumulating knowledge in stages to solve graph tasks more effectively. This study is motivated by recent advancements that hint at the possibility of leveraging iterative prompts to enhance LLM problem-solving. In particular, the CR framework presents an innovative approach, proposing that guiding models through iterative propositional and verification steps could enable LLMs to handle complex, structured problems more robustly.

Building on this insight, the current research seeks to investigate whether CR, initially a theoretical approach, can be adapted to graph-based problems as evaluated in the NLGraph benchmark. By applying CR to a structured graph problem set, this study aims to bridge the gap between iterative prompting theory and practical LLM reasoning in complex domains. The hope is to unlock new levels of performance and adaptability in LLMs when faced with recursive, interconnected reasoning tasks typical of graph theory.

1.2 Foundational Studies and Frameworks

1.2.1 NLGraph Benchmark

The NLGraph benchmark [8], introduced by Wang et al., serves as a comprehensive tool for evaluating LLMs on a range of graph-theoretic reasoning tasks. Unlike traditional benchmarks that assess LLM performance on linear or text-based reasoning, NLGraph is specifically designed to evaluate the ability of LLMs to handle complex, structured graph problems. This benchmark revealed a surprising brittleness in LLMs when applied to graph-theoretic tasks, highlighting that while models may perform reasonably well on simpler tasks, their performance often deteriorates as problems become more interconnected and demanding. The NLGraph study exposed these limitations by presenting LLMs with graph problems that require maintaining logical coherence across multiple reasoning steps, a challenge that conventional prompting techniques have struggled to address.

The NLGraph dataset encompasses a diverse set of graph reasoning categories, each aimed at testing a different aspect of graph-related problem-solving capabilities. The primary categories include:

- **Connectivity**: Tasks that involve determining whether a path exists between specified nodes in a graph, a fundamental aspect of graph theory that tests the model's basic comprehension of graph traversal.
- **Cycle Detection**: Problems that require identifying whether a cycle exists within the graph structure, posing a logical challenge that involves understanding graph connectivity and closed paths.
- **Flow**: This category focuses on calculating the maximum flow through a network, which tests the model's ability to handle resource allocation and optimization within a constrained graph structure.
- **Graph Neural Network (GNN) Simulation**: These tasks are designed to evaluate the model's ability to approximate behaviors typical of graph neural networks, such as node classification or edge prediction, which require understanding graph structure on a deeper level.
- **Hamiltonian Path**: This task involves determining whether a path exists that visits each node exactly once, challenging the model's ability to engage in combinatorial reasoning and handle complex, multi-step problem-solving.
- **Topology**: Tasks in this category test the model's understanding of graph hierarchy and structure, such as identifying levels or parent-child relationships within a directed acyclic graph (DAG), which requires recognizing graph orientation and layered relationships.

Each category within the NLGraph dataset is further divided by complexity level—*Easy*, *Medium*, and *Hard*—allowing for a progressive assessment of the model's generalization abilities as task difficulty increases. For instance, an easy task in connectivity might involve a small, sparsely connected graph, whereas a hard task could require identifying connectivity

within a larger, more densely connected structure. This stratified complexity approach provides a nuanced framework for evaluating LLMs across different levels of difficulty. However, due to time constraints, this research excludes the *Matching* and *Shortest Path* categories, allowing for a focused analysis on the remaining tasks that still capture a broad spectrum of graph reasoning challenges.

Overall, the NLGraph benchmark offers significant advantages as an evaluation framework. Its structured, tiered approach provides insights into both the strengths and limitations of LLMs in handling graph reasoning, offering a clear diagnostic of where traditional prompting methods fall short. By highlighting these limitations, the NLGraph benchmark underscores the necessity for advanced prompting strategies—such as cumulative reasoning —to guide LLMs through iterative problem-solving steps. This benchmark thus provides a solid foundation for developing and testing innovative prompting techniques aimed at enabling LLMs to navigate the complexities inherent in graph-theoretic reasoning.

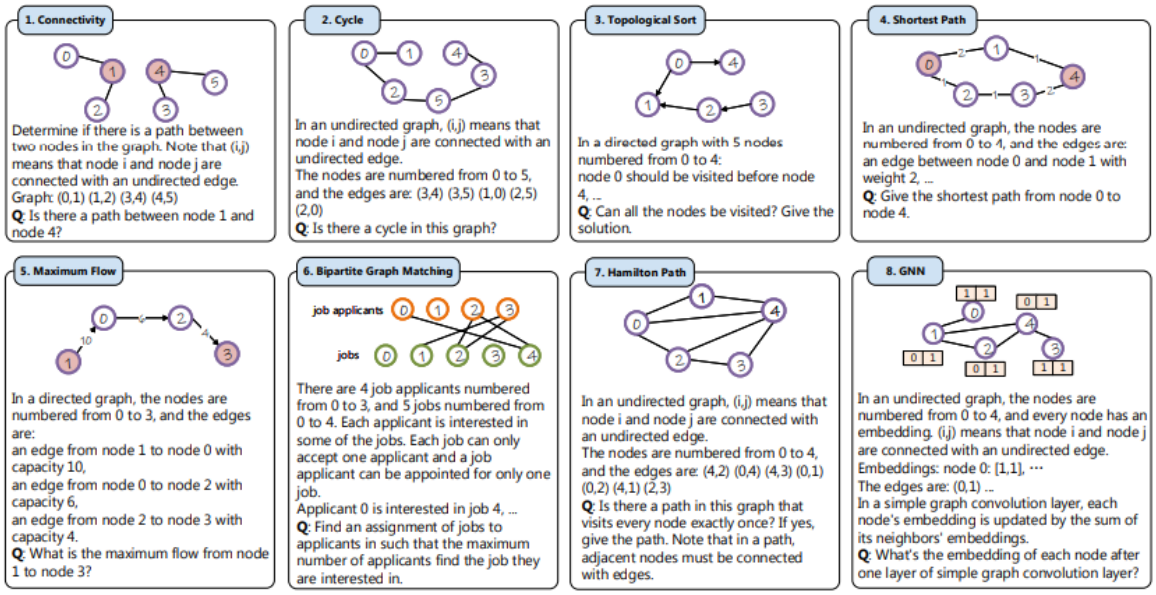

Figure 1.1: Overview of the NLGraph Benchmark, showing the variety of graph reasoning tasks and their complexity levels.

1.2.2 Cumulative Reasoning Framework

The Cumulative Reasoning (CR) framework, proposed by Zhang et al[10]., introduces an innovative approach to prompting LLMs by structuring complex tasks as iterative, multi-step reasoning processes. Unlike conventional single-step or linear prompting methods, CR operates within a Directed Acyclic Graph (DAG) structure, where each reasoning step builds on previously validated steps, forming a cumulative path toward a solution. This framework is particularly well-suited to tasks requiring logical consistency across multiple layers of reasoning, making it ideal for applications in structured domains, such as graph-theoretic problem-solving.

At the core of the CR framework are three specialized roles that facilitate the iterative reasoning cycle: the **Proposer**, the **Verifier**, and the **Reporter**. Each role contributes uniquely to the process, working collaboratively within the DAG structure to ensure accuracy and coherence in the final output:

- **Proposer**: The Proposer is responsible for generating potential solutions or intermediate reasoning steps based on the task's current context. Acting as the initiator, the Proposer introduces possible moves toward the solution, setting the stage for further validation. This role enables the model to iteratively explore different approaches, adapting its strategy as the reasoning path develops.
- **Verifier**: The Verifier evaluates each step generated by the Proposer, checking for logical consistency and accuracy. This role acts as a quality checkpoint within the reasoning cycle, preventing the accumulation of errors and ensuring that only validated steps are retained. The Verifier's function is critical in maintaining the logical integrity of the cumulative reasoning path, as it discards steps that do not meet established accuracy standards.
- **Reporter**: The Reporter compiles all verified steps into a cohesive final solution, signaling the conclusion of the reasoning process once a satisfactory answer is reached. This role is responsible for interpreting the completed reasoning chain, synthesizing the steps into a clear and structured output. The Reporter ensures that the cumulative steps form a coherent, logical solution that accurately addresses the task requirements.

Through this role-based, iterative cycle, the CR framework enables LLMs to engage in a recursive reasoning process where each validated step enhances the overall solution quality. The DAG structure used in CR allows for flexibility, as it can accommodate multiple validated steps branching off in different directions before converging into a final solution. This recursive, cumulative approach helps mitigate logical inconsistencies that may arise in complex tasks by validating each step before proceeding, ultimately promoting greater accuracy and coherence.

Zhang et al. demonstrated the effectiveness of the CR framework in various structured reasoning tasks, observing significant improvements in model performance, particularly on problems that require multi-layered, iterative reasoning. CR proved to be especially beneficial in tasks that demand a high degree of logical coherence across multiple stages, where traditional prompting techniques often fail to maintain consistent logical structure.

In this study, the CR framework serves as an inspiration and foundation for adapting cumulative reasoning to graph-theoretic tasks, particularly within the context of the NLGraph benchmark. However, instead of a direct implementation, the approach here involves adapting and modifying certain aspects of CR to better align with the specific demands and constraints of graph-based reasoning. For instance, graph-theoretic tasks often involve recursive relationships between nodes, where intermediate steps need to reference previous decisions and structural dependencies. The CR framework's iterative design, which inherently supports cumulative validation, is particularly advantageous in this context, as it can handle the layered dependencies typical in graph tasks.

The adaptation of CR in this research leverages its core principles—proposing, verifying, and reporting—but customizes the reasoning cycle to optimize it for graph-theoretic problem-solving. The adapted CR framework in this study is structured to work seamlessly within the NLGraph dataset, applying cumulative reasoning to evaluate model performance on a range of tasks, from connectivity and cycle detection to more complex challenges like Hamiltonian paths and graph neural network (GNN) simulations. By embedding the CR methodology within the NLGraph dataset, this research seeks to evaluate whether iterative reasoning can enhance LLM performance on graph-based tasks, focusing on recursive logic and validation processes that are essential for maintaining coherence in graph reasoning.

Further details on the implementation of this modified CR approach are discussed in the Methodology section, where the roles and iterative cycle are tailored specifically for graph reasoning. By combining the CR framework's role-based, DAG-driven structure with the challenging, structured tasks of the NLGraph benchmark, this study aims to investigate whether the cumulative reasoning approach can overcome some of the inherent limitations observed in traditional prompting methods. The hope is that this adapted CR framework will provide a more robust solution for LLMs, enabling them to tackle graph-related tasks with improved logical consistency, accuracy, and generalization.

1.3 Research Questions

- **How can CR prompting strategy be effectively designed for LLMs to handle graph-based reasoning tasks?**
  This question seeks to develop an adaptable CR framework tailored to meet the unique challenges of graph-theoretic problem-solving, as detailed in the methodology.
- **What is the performance difference compared to traditional prompting techniques, and why does CR achieve improvements?**
  This question focuses on evaluating performance gains of CR over traditional techniques and understanding the underlying reasons for these improvements, as shown in the analysis.

Chapter 2

Literature Review

2.1 Introduction to Prompting Techniques for LLMs

Prompting techniques have become essential in enhancing the reasoning capabilities of LLMs, especially for complex, multi-step tasks. Early approaches like zero-shot and few-shot prompting allowed LLMs to address basic questions with minimal task-specific training but struggled with intricate reasoning tasks requiring a structured, step-by-step approach.

To overcome these limitations, the **CoT prompting** technique was introduced, enabling LLMs to decompose problems into intermediate reasoning steps rather than producing direct answers. This method enhances model performance on multi-step problems by promoting logical consistency and mimicking human-like thought processes [9]. CoT provides a transparent pathway from problem to solution, making reasoning steps more interpretable.

Recent research has sought to refine and automate prompting strategies. Zhang et al. (2022) proposed an **Automatic Chain-of-Thought (Auto-CoT)** prompting method, designed to streamline the reasoning process by generating reasoning chains automatically. This reduces reliance on manual prompt engineering, making CoT more scalable. However, challenges remain in managing error propagation across reasoning steps, as early inaccuracies can affect final outcomes [9]. Auto-CoT underscores the importance of iterative validation within prompting, as errors in earlier stages may compromise later steps.

Domain-specific adaptations of CoT have also emerged. Cao et al. (2024) introduced a **Framework Chain-of-Thought (Framework CoT)** strategy tailored to systematic review screening in medical research, ensuring that reasoning aligns with domain-specific standards. This specialized adaptation highlights CoT's versatility, showing its adaptability for structured, domain-specific reasoning [2]. Framework CoT's success suggests that CoT could be further tailored for complex reasoning domains like graph theory, which demands systematic, multi-step logic.

Studies on prompting have further examined the role of template structure. Kim et al. (2023) evaluated multiple prompting strategies for natural language generation (NLG), showing that the effectiveness of prompting is highly context-dependent, with certain structures yielding better results for specific tasks [6]. This finding emphasizes the need to tailor prompts to the

unique demands of each domain, particularly in graph reasoning, where maintaining logical coherence and structural comprehension is critical.

Together, these studies illustrate the significant advancements achieved through CoT, Auto-CoT, and Framework CoT, while also highlighting their limitations in handling structured, multi-layered tasks where logical consistency is essential. These limitations point to the need for more sophisticated prompting techniques, such as **CR**, which introduces iterative validation to improve performance on tasks with high logical interdependency. The evolution of prompting strategies thus provides a foundation for developing CR frameworks tailored to complex graph reasoning tasks.

2.2 Challenges in Graph-Theoretic Reasoning with LLMs

Graph-theoretic reasoning poses unique challenges for LLMs, as it requires navigating recursive structures, interconnected nodes, and multi-step logical processes that differ from linear textual reasoning. Traditional LLMs, primarily optimized for language tasks, struggle to generalize these capabilities to graph reasoning, revealing substantial limitations in structured reasoning abilities.

To address these challenges, Jiang et al. (2023) proposed **StructGPT**, a framework that enhances LLM reasoning over structured data through specialized tool interfaces [5]. StructGPT combines iterative reading with structured data processing, guiding models through complex reasoning steps with the help of external tools. This approach allows LLMs to handle dependencies in graph-based problems more effectively but also highlights the limitations of LLMs when handling structured data independently.

Similarly, Sun et al. (2024) introduced the **THINK-ON-GRAPH (ToG)** framework, which integrates LLMs with knowledge graphs to facilitate dynamic, responsible reasoning across graph structures [7]. ToG uses beam search and adaptive exploration to enable LLMs to traverse graphs while maintaining logical consistency, thereby addressing the need for iterative exploration and path validation. By adjusting reasoning paths based on prior steps, ToG improves accuracy and coherence in multi-step tasks, underscoring the necessity of dynamic adaptability for effective graph reasoning.

Agrawal et al. (2023) explored LLM performance in structured graph reasoning through a set of benchmarks aimed at complex traversal tasks [1]. They introduced **PathCompare**, a prompting technique that evaluates the similarity of traversal paths within graphs. While PathCompare demonstrated some improvement, performance gains were limited on tasks requiring high levels of logical coherence, underscoring the need for even more advanced techniques for intricate graph structures.

These studies collectively highlight the formidable challenges that LLMs face in graph-theoretic reasoning. While LLMs are proficient in general language tasks, they lack the structured reasoning capabilities required for complex graph problems without augmentation. And there are many amazing steps forward but also reinforce the limitations of conventional LLM architectures in graph-based reasoning. This motivates the development of specialized prompting techniques, such as **CR**, to uphold logical consistency and manage recursive dependencies in graph reasoning tasks.

2.3 Evaluation and Benchmarking in Reasoning Tasks

Evaluating LLM reasoning capabilities is crucial to understanding model strengths and weaknesses, especially for tasks that extend beyond traditional NLP. Chang et al. (2024) provide a comprehensive survey on LLM evaluation methods, emphasizing the importance of evaluating task complexity, robustness, and ethical implications [3]. They highlight that while LLMs perform well on standard NLP tasks, they struggle with tasks that demand out-of-distribution reasoning and adaptability to novel situations. The authors advocate for a structured evaluation framework that assesses performance on both in-distribution and out-of-distribution data, evaluating model stability under unfamiliar conditions.

Collins et al. (2022) further support this view by introducing a benchmark designed to evaluate LLMs on out-of-distribution reasoning tasks [4]. Their benchmark includes domains like planning and causal explanation, which require flexible, structured thinking. The study reveals that while LLMs can replicate patterns, they struggle with complex reasoning, reinforcing the need for hybrid approaches that combine LLMs with symbolic reasoning modules. Collins et al. suggest that augmenting LLMs with structured reasoning modules could enhance robustness and adaptability, particularly for tasks that require multi-step logical consistency.

These findings underscore the need for refined and rigorous benchmarking methods that can capture the realistic demands of complex reasoning tasks, such as those found in graph theory. This study aims to explore **CR** within this context, using a framework that mirrors the complexities of graph reasoning to provide a more accurate measure of LLM performance in structured reasoning tasks.

2.4 Summary of Literature Review

There are significant advancements and ongoing challenges in prompting techniques and evaluation methods for LLMs, particularly in complex reasoning domains. Traditional prompting techniques have encountered difficulties in managing structured, multi-layered tasks. These limitations have led to the exploration of more advanced frameworks like **CR**, which as mentions times and times, offers iterative validation to address high logical interdependency.

Graph-theoretic reasoning remains especially challenging for LLMs, as shown by frameworks like **StructGPT**, **ToG**, and **PathCompare**, which highlight the inherent limitations of traditional LLMs in handling recursive and interdependent tasks without external augmentation. Evaluative research, such as that by Chang et al. and Collins et al., emphasizes the need for rigorous, task-specific benchmarks to fully understand model performance in structured reasoning contexts. This study builds upon these insights, aiming to develop a tailored CR framework to address the complexities of graph reasoning tasks and to provide a more robust evaluation of LLM capabilities.

Chapter 3

Methodology

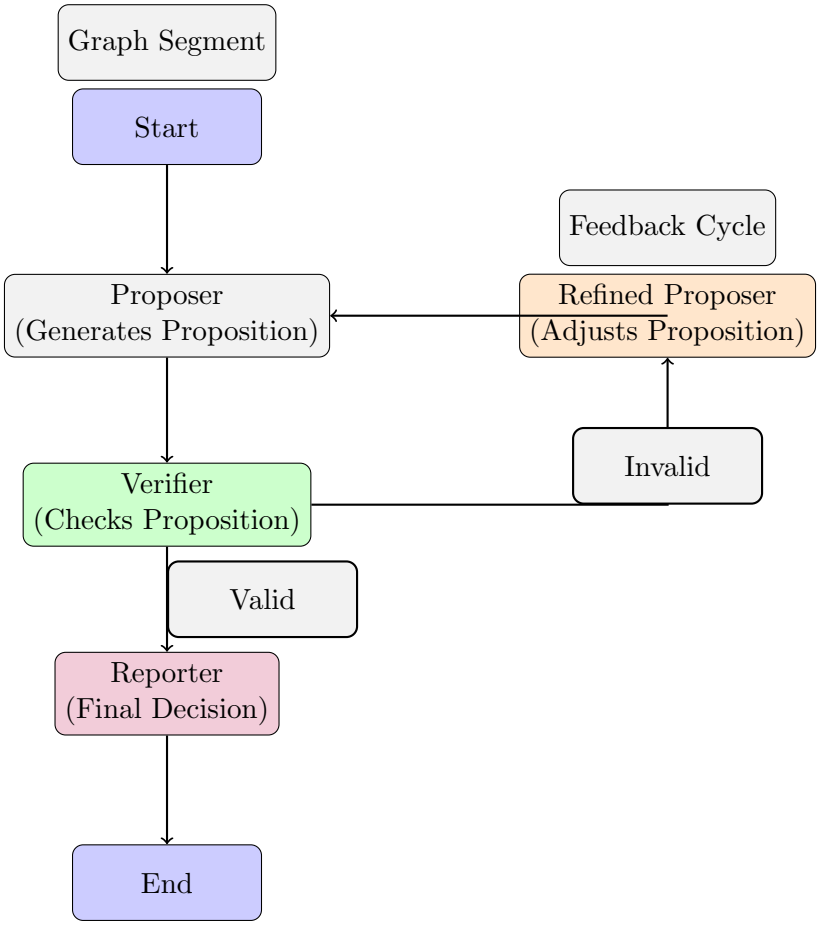

Figure 3.1: Workflow of the CR Framework

3.1 Overview and Challenges in Graph-Theoretic Reasoning with LLMs

This chapter outlines the methodology used to enhance the problem-solving capabilities of LLMs for complex graph-theoretic reasoning tasks through a tailored **CR** prompting strategy. Building on limitations identified in previous chapters, the approach adapts the CR framework to address the recursive, multi-layered nature of graph reasoning tasks as evaluated within the NLGraph benchmark. The methodology integrates dynamic prompt generation for each question type (QType) to ensure that the Proposer begins with a targeted prompt based on the task requirements. Each phase in the reasoning cycle—including proposition generation, verification, refinement, and reporting—is logged for traceability, ensuring logical consistency at every step. This iteration, combined with the feedback cycle, enables the model to handle dependencies across graph segments, thus enhancing cumulative understanding. The above figure (Figure 3.1) illustrates this process.

Graph-theoretic reasoning tasks present unique challenges for LLMs that traditional prompting methods, including standard CoT prompting, struggle to address effectively. The core difficulty lies in the recursive and dependency-driven nature of graph problems, which often require cumulative reasoning across multiple propositions and iterative validations. Unlike tasks with linear logic flows, graph problems demand an understanding of complex relationships, such as paths, cycles, and dependencies, which evolve as new segments of the graph are revealed.

The limitations of traditional prompting stem from the lacking the mechanism to retain and reference intermediate propositions or conclusions, each API call essentially operates in isolation. This absence of context forces each reasoning step to reconstruct prior conclusions, which can lead to redundant or logically inconsistent propositions.

To address these issues, the CR framework is adopted. CR incorporates an iterative refinement process where propositions are adjusted based on verifier feedback, fostering a stepwise accumulation of validated insights. This process is particularly suited for graph-based tasks, where the goal is to gradually build an accurate mental model of the graph's structure. By structuring prompts in a sequential and layered manner, the CR framework enables the model to handle dependencies more effectively, avoiding premature conclusions and maintaining logical coherence across steps. This adaptation was essential for capturing the recursive nature of graph-theoretic reasoning, setting CR apart as a suitable methodology for the NLGraph benchmark.

3.2 Adaptation of the Cumulative Reasoning Framework

3.2.1 Rationale for Adaptation

As discussed in the Introduction, traditional prompting techniques and even the original CR framework encounter challenges when applied directly to graph-theoretic reasoning due to the inherent complexity and recursive dependencies of such tasks. The original CR framework, while effective for certain structured reasoning problems, was not explicitly designed to handle

the multi-layered logic and iterative validations required in graph reasoning. Therefore, I propose an adapted version of the CR framework that modifies the roles and processes to better suit the specific demands of graph-based tasks.

3.2.2 Modified Roles and Iterative Process

The adapted CR framework preserves the foundational roles of **Proposer**, **Verifier**, and **Reporter**, but refines their functions and responsibilities to better suit the requirements of the NLGraph dataset. In this approach, I used a single universal file to manage the CR framework, while the specific prompt texts for each question type are organized in separate folders. Each question type contains a unique set of keywords, which are stored in a dictionary for easy retrieval and alignment with the relevant task, more details later on in the chapter.

- **Proposer**: Generates propositions based on incrementally revealed segments of the graph, focusing on observations and basic analysis of the current given graph without making final conclusions nor leading/implying towards a segmental assumption. This encourages the model to build the prompts by understanding the graph progressively, accommodating the recursive nature of graph structures.
- **Refined Proposer**: Introduced as an enhancement over the original framework, this role refines previous propositions when they fail verification, incorporating feedback to improve logical consistency. This is crucial for graph tasks where early errors can significantly impact subsequent reasoning.
- **Verifier**: Assesses the logical soundness of each proposition by comparing it against a locally constructed full graph segment. This process includes keyword detection, logical validation, and definition checking to ensure consistency with graph properties and predefined definitions. Through this workflow, the Verifier maintains logical coherence and alignment with the structural requirements of each graph reasoning task.
- **Reporter**: Compiles all verified propositions into a cohesive, refined proposition along with the original question. This consolidated information is used to reach a final conclusion regarding the graph's properties, such as the presence of a cycle or the connectivity between specific nodes.

3.2.3 Iterative Reasoning Cycle (Cumulative Reasoning Flow)

The reasoning process follows an iterative cycle in which graph segments are sequentially presented to the Proposer, who generates propositions based on each segment. These propositions are subsequently verified, and the cycle continues until all graph segments have been processed. The inclusion of the Refined Proposer role enables dynamic adjustments based on feedback from the Verifier, thereby enhancing the robustness and accuracy of the reasoning process. This flow ensures seamless coordination among the four primary roles, facilitating a cohesive and effective reasoning strategy.

3.3 Implementation Details

3.3.1 Model Configuration

For this study, I employed the ChatGPT-3.5 model- 0125, accessed via the Open AI API, as the primary engine for generating and validating responses within the cumulative reasoning framework. Configuring this model to perform optimally for structured reasoning tasks required careful tuning of several key parameters. There is also another reason due to version control, which will be further explained in chapter of Result analysis.

One such parameter was the temperature, set at 0.75. This value was chosen to strike an effective balance between response diversity and focused relevance. A higher temperature can lead to more varied responses, promoting creativity, but often at the cost of consistency. Conversely, a lower temperature yields more deterministic responses, but risks making the model overly rigid or repetitive. By setting the temperature at 0.75, I aimed to allow the model enough flexibility to explore reasoning paths creatively, while ensuring that each response remained grounded in logical coherence and pertinent to the question at hand.

In addition to temperature, I imposed a token limit of 500 tokens per response. This limit was implemented to control the length and focus of each output, ensuring that the model's responses were concise yet informative. Keeping responses within this token range helps prevent the model from drifting off-topic, which can occur when responses are too lengthy. It also encourages the model to provide clear and structured insights, particularly valuable in cumulative reasoning tasks where each response builds on previous steps. Furthermore, by constraining response length, I focused to maintain computational efficiency, reducing processing time and ensuring the model operates within the constraints of iterative reasoning.

3.3.2 Dataset Preparation

As mentioned, Since this approach will utilize the NLGraph benchmark, focusing on the following categories: *Connectivity*, *Cycle Detection*, *Flow*, *Graph Neural Network (GNN) Simulation*, *Hamiltonian Path*, and *Topology*. Each category provides tasks of varying difficulty levels—*Easy*, *Medium*, and *Hard*—allowing us to assess the generalization abilities of the CR framework across complexity gradients.

3.3.3 Dynamic Prompt Generation

Prompts were dynamically generated based on the question type (`QType`) and the specific graph segment being presented. Templates were crafted for each role—Proposer, Verifier, and Reporter—and were customized using Python string formatting to incorporate relevant details, such as the partial graph, hypothesis, and any prior propositions generated during the reasoning process. This approach ensured that each prompt was tailored to the context of the current reasoning step, allowing for accurate and role-specific guidance.

For instance, the prompt for the Proposer in a cycle detection task was designed to encourage detailed, logically sound propositions. An example prompt for the Proposer in the "cycle"

question type is shown below:

Proposer Prompt Example for Cycle Detection

You are a problem solver in graph theory. Based on the following partial graph, provide a proposition that will assist in understanding or identifying if there is a cycle in the given graph or not.
Your proposition should include:

- A clear and concise definition of a cycle, if not already provided: "A cycle is a path in a graph that starts and ends at the same node, passing through a sequence of distinct edges and nodes without revisiting any nodes, except the starting and ending node."
- Accurate statements of the connections for the provided nodes, matching exactly the partial graph presented.
- Logical analysis and observations based strictly on the given edges, aiming to identify conditions that indicate the presence or absence of a cycle.

Please ensure:

- Use clear and precise language.
- **If you can logically conclude that a cycle exists or does not exist based on the given edges, you may state this conclusion and provide your reasoning.**
- Do not include trailing commas in lists of nodes.
- Avoid logical fallacies and ensure that your reasoning is sound and valid.

Partial Graph: {`partial_graph`}
Please analyze the partial graph and provide observations or propositions based on the current information. **Do not make any final conclusions** about the presence or absence of a {`QType`}. Focus on exploring possible paths, connections, and patterns that may be relevant to identifying a {`QType`} in the future.
Your response should:

- Be clear and concise.
- Build upon previous propositions without repeating them verbatim.
- Avoid making definitive statements about the presence or absence of a {`QType`}.

Remember, the goal is to collaboratively build understanding without reaching a final decision at this stage.
Hypothesis: {`hypothesis`}

This structured prompt guides the Proposer in generating insightful propositions without reaching premature conclusions, ensuring that each contribution is logically consistent and

contributes incrementally to the overall reasoning process.

To accommodate the unique requirements of each role, separate prompt templates were designed for the Proposer, Verifier, and Reporter. For example:

- **Proposer Prompts**: Focused on exploring potential connections and logical implications based on the current graph segment. Proposers were instructed to avoid final conclusions and instead provide propositions that could be cumulatively built upon.
- **Verifier Prompts**: Directed the Verifier to assess the logical soundness of each proposition by comparing it with the existing graph structure and verifying consistency with the definitions specific to the question type.
- **Reporter Prompts**: Tasked with compiling validated propositions to present a final, cohesive statement regarding the graph's properties, such as the presence of a cycle or connectivity.

Each prompt template was customized dynamically through Python string formatting. Parameters such as {`partial_graph`}, {`hypothesis`}, and {`QType`} were replaced at runtime to ensure that the prompts were contextually relevant to the current task. This dynamic customization allowed for adaptability across different question types and graph structures, making the CR framework flexible and extensible for a variety of graph-theoretic reasoning tasks.

Overall, the dynamic prompt generation approach facilitated a structured yet adaptable methodology for guiding each role's contribution. By tailoring prompts specifically to the task and role, the process ensured that the model's responses were consistently aligned with the incremental reasoning requirements of the CR framework.

3.3.4 Modified Roles and Iterative Process

The adapted CR framework retains the fundamental roles of **Proposer**, **Verifier**, and **Reporter**, while adjusting their functions to meet the specific requirements of testing on the NLGraph dataset. This framework follows an iterative reasoning cycle where each role collaborates to incrementally analyze the graph, avoiding premature conclusions. Dynamic prompts are stored by question type and adapted to each task to maintain logical coherence across reasoning steps.

Proposer

The Proposer generates initial propositions based on a given partial graph and hypothesis, focusing on observations rather than definitive conclusions. This design aligns with the incremental nature of graph exploration and enables cumulative reasoning. Each prompt is customized based on the graph segment and question type.

Code Example: Proposer Function

```
def Proposer(partial_graph, hypothesis, QType):
    prompt = create_prompt(partial_graph, hypothesis, QType)
    response = openai.chat.completions.create(model, prompt)
    return response
```

Here, `create_prompt` dynamically generates the prompt based on the partial graph and question type, allowing for targeted exploration without final conclusions.

Refined Proposer

If a proposition fails verification, the Refined Proposer re-evaluates and adjusts the proposition by incorporating feedback from the Verifier. This feedback loop ensures logical consistency across iterative steps, progressively refining the model's understanding.

Code Example: Refined Proposer Function

```
def Refined_Proposer(partial_graph, hypothesis,
                     last_failed_proposition, QType):
    refined_prompt =
            modify_prompt_with_feedback
                (last_failed_proposition)
    response = openai.chat.completions.create
            (model, refined_prompt)
    return response
```

The function `modify_prompt_with_feedback` integrates feedback to refine the proposition, enhancing logical rigor without making final conclusions.

Verifier

The Verifier assesses each proposition for logical soundness and consistency with the graph's properties. It checks if the proposition aligns with predefined graph rules, relevant definitions, and logical requirements specific to the question type. This step prevents the accumulation of logical errors by verifying each reasoning step independently.

Code Example: Verifier Function

```
def Verifier(proposition, premises, QType):
    # Parse edges and validate logical consistency
    if edges_in_proposition match edges_in_premises
        and definitions_valid:
        return 'True'
    else:
        return 'False'
```

The Verifier checks that the proposition's edges align with the premises and verifies the definitions. If inconsistencies are detected, the proposition is flagged for refinement.

Reporter

The Reporter compiles all verified propositions and synthesizes them into a final conclusion regarding the graph's property in question, such as cycle detection or connectivity. It provides a summary based on accumulated propositions and outputs a conclusive answer, starting with "Yes" or "No" and followed by a concise explanation.

Code Example: Reporter Function

```
def Reporter(propositions, original_problem, QType):
    summary = summarize_propositions(propositions)
    final_response = openai.chat.completions.create
    (model, summary)
    return final_response
```

The `summarize_propositions` function aggregates verified propositions, supporting a final response that is both logically consistent and comprehensive.

3.3.5 Iterative Reasoning Cycle Implementation

The cumulative reasoning flow orchestrates the iterative reasoning process in the framework, where graph segments are presented to the Proposer for proposition generation. Each generated proposition is subsequently verified, and if it fails verification, the Refined Proposer generates a revised proposition based on Verifier feedback. This iterative cycle continues until all graph segments have been processed, ensuring that reasoning remains logically consistent and robust across each step.

The iterative nature of this approach is designed to support cumulative understanding, as each verified proposition builds on previous insights. The Reporter then compiles the verified propositions into a final decision regarding the graph's property in question.

Code Example: Cumulative Reasoning Flow

```
def cumulative_reasoning_flow(graph_segments,
                              hypothesis, QType):
    # Iterate over each segment of the graph
    for segment in graph_segments:
        proposition = Proposer(segment, hypothesis, QType)

        # Verify each proposition, refining if necessary
        if Verifier(proposition, premises, QType) == 'False':
            proposition = Refined_Proposer(segment,
                                           hypothesis,
                                           proposition,
                                           QType)

        # Accumulate verified propositions
        add_to_verified_propositions(proposition)

    # Make final decision based on all verified propositions
    return Reporter(verified_propositions,
                    original_problem,
                    QType)
```

In this Code Example, `graph_segments` represents the segments of the graph presented incrementally, while `QType` defines the question type (e.g., cycle detection). This function orchestrates the collaborative effort of the Proposer, Refined Proposer, Verifier, and Reporter, establishing a coherent reasoning process for each graph-based problem.

3.3.6 Evaluation Strategy and Metrics

To rigorously assess the performance of the CR framework, I implemented an evaluation strategy tailored to align with the goals of cumulative reasoning. My primary metric was the accuracy of the assistant's final decision for each graph, which reflects the effectiveness of the model in reaching a correct conclusion through iterative reasoning. Unlike traditional step-by-step assessments, this approach focuses on the end-to-end reasoning outcome, emphasizing the accuracy of the final decision rather than isolated intermediate steps.

Evaluation Metric: Correctness Determination

```
def determine_correctness_by_graph_index(graph_index,
                                         final_decision,
                                         total_graphs):
    if graph_index < total_graphs / 2:
        expected_answer = "no"
    else:
        expected_answer = "yes"
    is_correct =
        final_decision.lower() == expected_answer.lower()
    return is_correct, expected_answer
```

My evaluation strategy diverges from the original NLGraph benchmark, which relied on an index-based validation system with predefined response indices for each graph instance. Instead, I adopted a binary validation approach, wherein:

- The first half of each dataset subset was designated as "false" (e.g., no cycle or no connectivity).
- The second half was designated as "true" (e.g., cycle exists or connectivity present).

This binary division serves several purposes:

- **Simplified Validation Process**: By avoiding predefined response indices, this method allows for a more streamlined validation, reducing complexity and focusing on cumulative reasoning outcomes.
- **Alignment with Cumulative Reasoning Objectives**: This approach emphasizes the quality of the final output as a result of iterative reasoning, rather than evaluating each individual step in isolation.
- **Logical Consistency Emphasis**: The strategy allows us to assess the model's logical consistency by verifying that each proposition contributes coherently toward the final decision.

By leveraging this binary validation strategy, I can evaluate the model's cumulative reasoning abilities more effectively. Rather than testing individual reasoning steps for accuracy, this approach ensures that the model's responses are logically sound and contribute to an accurate cumulative output.

3.3.7 Outcome Assessment: Final Decision and Logical Coherence

In addition to basic accuracy, the evaluation framework considers two complementary aspects of performance:

- **Accuracy of the Final Decision**: The final answer is assessed based on the binary division, allowing us to determine whether the CR framework leads to correct conclusions by aggregating validated propositions.
- **Coherence of Intermediate Steps**: I examine whether each intermediate proposition logically supports the final decision. This coherence check ensures that all reasoning steps build on each other without contradictions, enhancing the robustness of the model's cumulative reasoning capabilities.

By focusing on both accuracy and coherence, this strategy provides a holistic evaluation of the CR framework's effectiveness in navigating complex, multi-step reasoning tasks. It also highlights areas where logical inconsistencies or premature conclusions may arise, offering insights into potential refinements for cumulative reasoning in graph-based tasks.

3.3.8 Graph Segment Presentation Strategy

To manage the inherent complexity of graph-based reasoning tasks, I implemented a strategy to divide graphs into smaller, manageable segments. This segmented approach allows the Proposer to focus on one part of the graph at a time, building a progressive understanding of the structure rather than being overwhelmed by the entire graph at once. This incremental exposure aligns with the cumulative nature of graph exploration, facilitating stepwise reasoning and iterative proposition development.

Code Example: Graph Segment Presentation Strategy

```
def cumulative_reasoning_flow(edges, n, hypothesis,
                              QType, **kwargs):
    # Divide sorted edges into segments
    edge_segments = [sorted_edges[i : i + segment_size]
                     for i in range
                        (0, len(sorted_edges), segment_size)]
    for current_edges in edge_segments:
        # Generate and verify propositions for each segment
```

This segmentation strategy ensures that each segment of the graph is individually analyzed, allowing cumulative reasoning to unfold progressively and enhancing the model's ability to maintain logical coherence across the entire reasoning process.

3.3.9 Question Type Customization

Each question type in this study required unique handling based on the specific properties of the task (e.g., cycle detection, connectivity). To address this, I created a dictionary-based customization strategy where each question type entry contains relevant parameters and instructions to guide the model's reasoning. This dictionary includes:

- **Standard Definition**: A consistent definition of key terms to ensure clarity and consistency across propositions.
- **Definition Keywords**: Keywords used to identify definitions within propositions, ensuring that the model appropriately references essential concepts.
- **Conclusion Phrases**: Phrases used to detect claims within propositions, enabling identification of conclusions.
- **Logical Support Function**: Specific functions tailored to validate the logical accuracy of conclusions within each task context.
- **Additional Instructions**: Tailored guidelines for the Proposer, providing role-specific instructions aligned with the question type.

This structure allows the model to dynamically adjust its reasoning approach based on the question type, ensuring a flexible yet consistent methodology that adapts to the specific demands of each graph-theoretic task.

3.3.10 Logical Support Functions

Custom logical support functions were implemented for each question type to validate the consistency and logical accuracy of propositions. Each function is named in the format `is_QTYPE_logically_supported` to clearly identify its relevance to a specific graph property. For instance, the `is_cycle_logically_supported` function verifies if a claim about the presence or absence of a cycle within a graph aligns with the actual structure of the graph.

Code Example: Logical Support Function for Cycle Detection

```
def is_cycle_logically_supported(G, claimed_conclusion):
    """
    Checks if the claimed conclusion about
    cycles is logically supported.
    Parameters:
        G (networkx.Graph): The graph built from premises.
        claimed_conclusion (str): 'present',
                                  'absent',
                                  or 'uncertain'.
    Returns:
        bool: True if the conclusion is
        logically supported, False otherwise.
    """
    has_cycle = len(nx.cycle_basis(G)) > 0

    if claimed_conclusion == 'present':
        return has_cycle
    elif claimed_conclusion == 'absent':
        return not has_cycle
    else:
        return True  # Accept if uncertain
```

The purpose of these logical support functions is to cross-check the propositions generated by the Proposer against the structure of a locally created graph representation. For example, in cycle detection tasks, I build a graph from the provided premises and use `nx.cycle_basis` to determine if any cycles are present. The function then evaluates the claimed conclusion (e.g., "cycle is present" or "cycle is absent") to ensure it logically aligns with the graph structure. This step helps maintain the model's logical rigor by preventing conclusions that contradict the underlying graph data.

3.3.11 Handling of Definitions and Claims

To maintain logical coherence across propositions, helper functions were developed to systematically extract definitions and identify claims within each proposition. These functions ensure that each proposition follows the required logical structure, facilitating consistent reasoning and preventing premature or unsupported conclusions. This setup enables the model to progressively refine its understanding without deviating from established definitions.

Code Example: Helper Functions for Definitions and Claims

```
def extract_definitions(proposition, definition_keywords):
    # Extracts sentences containing definitions
    # based on provided keywords
def infer_claim_from_proposition(proposition,
                                 conclusion_phrases):
    # Identifies and extracts claims within a
    # proposition based on keywords
```

By ensuring that definitions are explicitly referenced and claims are identified, these functions reinforce the model's adherence to logical structure and prevent it from making unsupported conclusions. This structured approach to handling definitions and claims is integral to the cumulative reasoning framework, as it ensures that each reasoning step builds on a foundation of clear, validated premises.

3.3.12 Graph Data Processing

To facilitate structured reasoning, graphs were sourced from the NLGraph dataset and processed in a manner that allowed for gradual complexity exposure. The dataset was categorized by difficulty level—`easy`, `medium`, and `hard`—with each category comprising a different number of graphs. This stratification enabled the model to demonstrate its reasoning capabilities across a spectrum of complexity, from sparse, simple graphs to densely connected, intricate structures. The counts of graphs per difficulty level are as follows:

Graph Count by Difficulty Level

```
graph_counts = { "easy": 150, "medium": 600, "hard": 400}
```

This approach allowed us to systematically evaluate model performance as the complexity of the graph structures increased. By adjusting the number of graphs based on difficulty, I was able to observe the model's scalability and resilience in handling progressively challenging graph reasoning tasks.

3.3.13 Conclusion

In conclusion,this is a structured and robust approach to enhancing large language model reasoning within complex graph-theoretic tasks. By adapting the cumulative reasoning framework specifically for recursive, multi-layered graph analysis, I achieve a refined process that emphasizes logical coherence and progressive understanding. This tailored approach sets a solid foundation for exploring and expanding the capabilities of LLMs in structured reasoning domains.

Chapter 4

Experiments, Results, and Analysis

4.1 Comparative Baseline Analysis: NLGraph Study vs. Current Experiment

As mentioned previously, to maintain a controlled comparison, I closely followed the experimental setup from the NLGraph study. The original NLGraph paper conducted its evaluations using the `gpt-3.5-turbo-0125` model. Although newer and potentially improved versions of Open AI's large language models are now available, I intentionally chose to use the same model for this study.

In the original NLGraph benchmark, a range of prompting techniques were evaluated. However, only Chain-of-Thought CoT prompting was consistently applied across all question types—Connectivity, Cycle, Topological, Flow and Hamilton, these are the most suitable baseline for this study. Table 4.2 presents the CoT results for each question type and difficulty level, serving as a point of comparison for the adapted CR framework. It is important to note that the results presented here were calculated directly from the raw logging files of their experiment. In the original NLGraph study, this level of detail in performance results was not fully disclosed, as the focus shifted toward enhancing model performance through advanced techniques such as Build-a-Graph Prompting (BAG) and Algorithmic Prompting. Consequently, specific baseline scores were omitted, necessitating our own data collection for a comprehensive baseline comparison.

In addition, difficulty levels in specific question types (e.g., "Hard" for Topological) are not represented in the NLGraph study due to the absence of corresponding test cases or not findable in the logging files, resulting in some "not applicable" scores in Table 4.2.

Lastly, due to time constraints, I only performed five test runs for each question type at each difficulty level. The results represent the average score across these five runs, providing a balanced view of the model's performance with CR prompting, enables us to capture a stable performance metric while acknowledging the inherent variability in the responses.

Table 4.1: Baseline Performance of CoT Prompting on the NLGraph Benchmark

Question Type	**Difficulty**	**CoT Score (%)**
Connectivity	Easy	81.8
	Medium	82.17
	Hard	77.21
Cycle	Easy	51
	Medium	47.1
	Hard	45
Topological	Easy	54.4
	Medium	0.9
	Hard	Not Applicable
Flow	Easy	10.0
	Medium	Not Applicable
	Hard	4.0
Hamilton	Easy	40.0
	Medium	Not Applicable
	Hard	8.0

Table 4.2: Cumulative Reasoning Prompting on the NLGraph Benchmark

Question Type	**Difficulty**	**CoT Score (%)**
Connectivity	Easy	83.6
	Medium	82.4
	Hard	80.2
Cycle	Easy	58.4
	Medium	53.2
	Hard	51
Topological	Easy	57.8
	Medium	4.4
	Hard	Not Applicable
Flow	Easy	14.4
	Medium	Not Applicable
	Hard	3.8
Hamilton	Easy	42.6
	Medium	Not Applicable
	Hard	7.8

4.2 Analysis of CR Prompting Performance

Across the evaluated question types, CR prompting demonstrates varying degrees of success, with noticeable performance boosts in tasks involving Connectivity and Cycle detection, as shown in Table 4.2. These tasks benefit from CR's iterative and cumulative approach, which allows the model to build on previous reasoning steps. In Connectivity tasks, the structured prompt sequence in CR effectively guides the model to analyze node relationships progressively, resulting in higher accuracy compared to standard CoT prompting. Similarly, for Cycle detection, CR's iterative validation of paths and connections aligns well with the task's logical requirements, leading to more consistent and accurate outputs.

4.2.1 Performance on Simple vs. Complex Tasks

For simpler tasks, such as basic Connectivity or Cycle detection at lower difficulty levels, CR prompting outperforms CoT by a significant margin. This suggests that CR's structured approach to reinforcing intermediate steps is highly beneficial for tasks that require stepwise reasoning but lack deep dependency chains. The model's ability to revisit and refine propositions is particularly valuable in these cases, as it minimizes logical errors and strengthens the accuracy of cumulative insights.

However, the results for more complex tasks, such as Flow and Hamiltonian Path, reveal certain limitations of CR prompting. These tasks involve intricate dependencies, including path optimization and capacity constraints, which are challenging for LLMs without native memory or contextual continuity across API calls. The marginal declines in CR performance for these tasks imply that, for highly dependent or optimization-based tasks, additional assistance may be required. This could include integrating memory mechanisms or providing auxiliary guidance to help the model retain crucial intermediate insights.

4.2.2 Detailed Analysis of CoT vs. CR Prompting on all graph tasks

Connectivity Tasks

For connectivity-related tasks, the performance differences between CoT and CR prompting reveal both consistent patterns and subtle improvements. CoT scores are 81.8%, 82.17%, and 77.21% for Easy, Medium, and Hard difficulty levels, respectively, indicating a slight decrease in accuracy as task complexity increases. In comparison, the CR prompting approach achieves 83.6%, 82.4%, and 80.2% across the same difficulty levels, representing a noticeable improvement at the Hard level (from 77.21% to 80.2%). This increase may be attributed to CR prompting's iterative feedback mechanism, which allows for refined understanding of complex graph structures. The results indicate that CR prompting is particularly effective for connectivity tasks involving higher difficulty, where cumulative insights likely play a significant role.

Cycle Detection Tasks

Cycle detection shows modest gains with CR prompting, particularly at the Easy level. CoT scores are recorded at 51%, 47.1%, and 45% for Easy, Medium, and Hard levels, respectively, displaying a clear downward trend as task complexity increases. Under CR prompting, performance rises to 58.4% for Easy, 53.2% for Medium, and maintains at 51% for Hard tasks. These improvements (7.4% on Easy, 6.1% on Medium, and 6% on Hard) suggest that CR prompting's incremental verification process enhances accuracy in detecting cycles, particularly in simpler graphs. The iterative nature of CR prompting appears to assist the model in revisiting propositions, which is beneficial for cycle detection tasks that require validation of repeating paths and dependencies.

Topological Tasks

Topological tasks display slight improvements with CR prompting at the Easy level. For CoT, the scores are 54.4% for Easy and 0.9% for Medium, with no score available for Hard due to the original paper not providing results for this difficulty. CR prompting scores are 57.8% for Easy and 4.4% for Medium, representing an increase of 3.4% and 3.5%, respectively. These gains, though modest, are indicative of CR prompting's ability to systematically verify node ordering and dependency relations, which is crucial for topological tasks. This may suggest that the CR framework, by breaking down propositions and incorporating feedback cycles, offers enhanced validation mechanisms for tasks that rely on structural properties.

Flow Tasks

For flow tasks, CR prompting does not demonstrate substantial gains and even shows a minor decrease in the Hard difficulty level. CoT achieved 10% on Easy and 4.0% on Hard, while CR prompting results in 14.4% for Easy but drops to 3.8% for Hard. The Easy level shows an increase of 4.4%, which may indicate that the CR approach supports basic flow analysis in simpler graph structures. However, the decrease on Hard level (-0.2%) suggests that CR's iterative framework may be less effective in scenarios requiring complex capacity and path evaluations, which are central to flow analysis in dense graphs. This could indicate that further refinement in CR's handling of flow-specific tasks is necessary to handle intricate capacity constraints and flow optimizations effectively.

Hamiltonian Path Tasks

Hamiltonian path tasks, known for their computational complexity, show mixed results. CoT prompting scores 40% for Easy and 8% for Hard, with no score available for Medium. CR prompting slightly improves performance to 42.6% for Easy but shows a minimal decrease to 7.8% for Hard. The improvement at the Easy level suggests that CR prompting aids in validating sequences in simpler Hamiltonian path instances. However, the decrease at the Hard level suggests that for highly complex path-finding tasks, CR prompting may not provide significant benefits over CoT. This outcome highlights the potential limitations of

CR prompting when dealing with path-specific tasks that involve complex backtracking and sequence validations.

Overall Observations

Across the evaluated question types, CR prompting demonstrates a generally positive impact on performance, with clear improvements in tasks that align well with cumulative reasoning. These results suggest that CR prompting is particularly effective in facilitating logical consistency and accuracy in tasks that benefit from iterative validation and structured reasoning.

- **Performance on Easier Tasks:** For relatively straightforward tasks such as Connectivity and Cycle detection, CR prompting shows notable improvements over CoT prompting. The structured, iterative refinement process of CR prompting appears well-suited for these types of problems, where stepwise proposition validation supports the model's ability to build upon previous insights. The higher scores observed for Connectivity and Cycle detection indicate that CR prompting helps the model navigate logical dependencies and validate propositions cumulatively, leading to more accurate outcomes.
- **Performance on More Complex Tasks:** In contrast, for more challenging tasks like Flow and Hamiltonian path detection, the performance gains from CR prompting are less pronounced. These tasks often involve intricate dependencies and path optimizations, which demand a more nuanced understanding of graph structures and relationships. The marginal declines observed in these areas suggest that, for complex and path-dependent tasks, CR prompting alone may not be sufficient to fully capture the task requirements. This finding indicates that LLMs may need additional support—potentially through specialized prompts or augmented strategies—to effectively tackle tasks with high interdependency and optimization demands.

In summary, CR prompting demonstrates a clear advantage for simpler graph-based reasoning tasks, such as connectivity and cycle detection, where its stepwise, cumulative structure is well-aligned with task requirements. However, for more complex tasks involving multiple layers of dependency and optimization, CR prompting shows limitations, pointing to the need for further refinement. My results highlight both the adaptive potential of CR prompting for structured reasoning tasks and the importance of task-specific modifications to maximize its effectiveness across diverse types of graph-theoretic challenges.

4.3 Key Findings and Limitations

4.3.1 Strengths

The CR prompting framework demonstrated several notable strengths:

- **Improved Performance through Structured Prompting:** One of the primary strengths of the CR prompting framework is that it achieves performance improvements simply by refining prompt structure. By structuring prompts in a more logical, stepwise manner, I enable large language models (LLMs) to handle complex, multi-step reasoning tasks with greater accuracy. This performance boost is achieved without introducing additional algorithms, relying instead on a more effective method of interaction with the LLM. This insight suggests that structured prompt refinement alone can yield measurable benefits, opening up possibilities for enhancing LLM performance in other similarly complex domains.
- **Enhanced Interpretability of Intermediate Reasoning Steps:** The iterative nature of CR prompting not only allows for better accuracy but also enhances the interpretability of the reasoning process. By breaking down complex tasks into smaller, verifiable propositions, each step in the reasoning cycle becomes more transparent and understandable. This approach allows us to trace the logic behind each proposition, making it easier to analyze where and why errors occur, which is valuable for debugging and refining LLM-driven systems.

4.3.2 Limitations

Despite its strengths, the CR prompting framework exhibits certain limitations:

- **Lack of Contextual Memory in API Calls:** The OpenAI API does not maintain a history across API calls, meaning that each request operates independently of previous ones. In this project, this limitation necessitates rolling out all prior propositions for each new prompt, which can result in very long prompts. This not only increases the length and complexity of the prompts but also complicates debugging, as each problem may involve a large number of API calls.
- **High Computational and Time Costs:** The iterative nature of CR prompting requires multiple API calls for each reasoning step. Each iteration includes calls to generate, verify, and refine propositions, which collectively add significant time to each run. Consequently, the CR framework has high computational and time demands, making it less efficient for real-time or large-scale applications. This limitation highlights the need for optimization, especially if CR prompting is to be applied to more extensive datasets or integrated into time-sensitive systems.

In summary, while the CR prompting framework demonstrates potential for enhancing LLMs in complex problem-solving tasks, the lack of memory across API calls and the associated time and computational demands remain key challenges. Future work could explore methods to mitigate these limitations, such as developing strategies to maintain context across API calls or optimizing the framework to reduce the number of iterations required.

4.4 Future Directions

The findings from this study illustrate both the promise and the limitations of CR prompting in enhancing LLM performance on graph-theoretic tasks. While the CR prompting approach offers notable improvements in handling structured, cumulative reasoning tasks—particularly for problems involving logical consistency and stepwise validation—there remain areas where further advancements are required to fully leverage the potential of LLMs in complex reasoning domains. In light of these observations, there are several potential directions for future work:

- **Integrating Memory Mechanisms for Multi-Step Reasoning:** One limitation of the current approach is the lack of memory persistence across API calls, requiring the model to re-process prior propositions with each new prompt. Implementing a memory mechanism—either through external memory augmentation or through a framework that allows context persistence—could streamline reasoning processes and improve efficiency. Such an approach could reduce redundancy in iterative tasks, providing a more seamless and cohesive reasoning flow for problems that involve multi-step dependencies.
- **Developing Optimization-Driven Prompting Techniques:** Certain graph-theoretic tasks, such as flow optimization and Hamiltonian path problems, involve intrinsic optimization challenges that are difficult for LLMs to interpret solely through sequential reasoning. Developing a CR framework that explicitly incorporates optimization principles—perhaps through guidance on heuristics or additional constraints—could improve model performance on these tasks. This could involve integrating mathematical reasoning or structured feedback loops that encourage the model to explore and refine solution paths effectively.
- **Evaluating Across Diverse LLM Architectures and Newer Models:** While this study controlled for model consistency by utilizing the GPT-3.5-turbo-0125 model, future research could expand to evaluate CR prompting across various LLM architectures, including state-of-the-art models like GPT-4 and other multimodal or specialized reasoning models.
- **Investigating Error Correction and Feedback Mechanisms:** Observations from this study highlight the need for robust error correction mechanisms in cumulative reasoning, especially for tasks where individual mistakes can propagate through multiple iterations. Future work could incorporate real-time feedback or error-detection layers within the CR prompting cycle.

In summary, while CR prompting has proven to be a valuable tool for improving LLM performance on structured reasoning tasks, it is clear that further development is needed to enhance its applicability and efficiency in more complex and specialized domains. Future research focused on adaptive prompting, memory integration, and task-specific optimization could significantly enhance the capabilities of LLMs for complex reasoning, paving the way for more sophisticated applications of LLMs in graph-theoretic problem-solving and beyond. The findings of this study serve as a foundation, illustrating the vast potential of CR prompting while also highlighting the pathways for innovation in the evolving field of LLM-based reasoning.

Chapter 5

Conclusion

This study successfully adapts the Cumulative Reasoning prompting strategy to enhance large language models' performance on graph-theoretic reasoning tasks within the NLGraph benchmark. By tailoring CR to address the recursive and multi-layered nature of graph problems, significant improvements were observed, particularly in tasks like connectivity and cycle detection. The iterative approach not only increased accuracy but also enhanced the interpret ability of the reasoning process. However, limitations were noted in more complex tasks such as flow analysis and Hamiltonian path detection, indicating areas for further refinement. Future work could focus on integrating memory mechanisms and optimizing computational efficiency to address these challenges. Overall, the adapted CR framework demonstrates promising potential for advancing LLM capabilities in complex reasoning domains without modifying the underlying model architecture or parameters.

Appendix A

Appendix A: Link to all works

The full source code for this thesis project can be found on GitHub. You can access it via the following link:

https://github.com/camtrca/MT2024_EH_SJ

Bibliography

[1] P. Agrawal, S. Karamcheti, A. Narayan-Chen, and D. Jurafsky. Can llms perform structured graph reasoning tasks? *arXiv preprint arXiv:2402.01805*, 2023.

[2] J. Cao, P. Liu, and T. Chen. Prompting is all you need: Llms for systematic review screening. *bioRxiv*, 2024.

[3] L. Chang, Y. Zhang, and M. Liu. A survey on evaluation methods for large language models in complex reasoning tasks. *Journal of Artificial Intelligence Research*, 58:1–34, 2024.

[4] P. Collins, A. Mahajan, and D. Kumar. Benchmarking large language models on out-of-distribution reasoning tasks. *arXiv preprint arXiv:2205.05718*, 2022.

[5] J. Jiang, Y. Zhang, Q. Wei, and J. Zhang. Structgpt: A general framework for large language model to reason over structured data. *arXiv preprint arXiv:2305.09645*, 2023.

[6] H. Kim, J. Lee, and K. Song. Which is better? exploring prompting strategy for llm-based metrics. *arXiv preprint arXiv:2311.03754*, 2023.

[7] J. Sun, T. Wu, X. Wang, Y. Han, and J. Zhang. Think-on-graph: Deep and responsible reasoning of large language models on knowledge graphs. *arXiv preprint arXiv:2307.07697*, 2024.

[8] H. Wang, S. Feng, T. He, Z. Tan, X. Han, and Y. Tsvetkov. Can language models solve graph problems in natural language? *NeurIPS*, 2023. 37th Conference on Neural Information Processing Systems.

[9] Y. Zhang, J. Yang, Y. Yuan, and A. C.-C. Yao. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*, 2022.

[10] Y. Zhang, J. Yang, Y. Yuan, and A. C.-C. Yao. Cumulative reasoning with large language models. *arXiv preprint arXiv:2308.04371*, 2023.