Graduation Thesis

Implementing Tree-of-Thought Prompting on Llama, the smaller parameter Large Language Model

July 2024

Department of Interdisciplinary Sciences

International Program on Environmental Sciences

08224524

Sheldon Feiyu Zhang

Abstract

Implementing Tree-of-Thought Prompting on Llama, the smaller parameter Large Language Model

Department of Interdisciplinary Sciences

International Program on Environmental Sciences

08224524

Sheldon Feiyu Zhang

We investigate the effectiveness of the Tree-of-Thought (ToT) method for enhancing long-range reasoning tasks in smaller parameter large language models (LLMs). While ToT has shown impressive results with models like GPT-4, it remains unclear how well this method scales down to smaller models. To address this, we conducted experiments using Llama2-7b and Llama3-8b on tasks including the Game of 24, Sudoku, and creative writing, comparing ToT with standard input-output (IO) and Chain-of-Thought (CoT) prompting strategies. Our results indicate a significant decline in performance for the ToT method with smaller models, similar to the decline observed with CoT. Specifically, ToT's success rate dropped to 0% with Llama2-7b for the Game of 24, compared to a 74% success rate with GPT-4. These findings suggest that while ToT is effective with large parameter models, it does not enhance performance in smaller models and highlights the need for further research into scalable prompting strategies for LLMs with limited parameters.

You can find more about our codes and experiment results on:

https://github.com/sheldonfeiyuzhang/PEAKGraduationProject

Table of Contents

1. Introduction

The advent of large language models (LLMs) has revolutionized the field of natural language processing, enabling machines to generate text and perform tasks that require short-range reasoning with remarkable accuracy. A year ago, the Tree-of-Thought (ToT) prompting method has been proposed to enhance the performance of LLMs in long-range tasks. The method involves representing abstract concepts as nodes in a tree-like structure, and using the LLM to generate text that describes the relationships between these nodes. This allows the LLM to explicitly represent and manipulate abstract concepts, and to perform long-range reasoning tasks more effectively.

However, the impressive performance of ToT comes at a great cost: token consumption is significantly higher than with zero-shot or few-shot prompts, which leads to significant power and calculation costs. Moreover, previous research has shown that the Chain-of-Thought (CoT) method, which also relies on step-by-step logic, experiences a significant decline in accuracy when applied to smaller parameter language models. The study by Wei et al. [15] found that the CoT prompting method resulted in an accuracy similar to standard prompting in various benchmarks when the model has a parameter of less than 10 billion. This decay in performance is attributed to the limited capacity of smaller models to capture complex contextual relationships between tokens, leading to a reduced ability to generate coherent and accurate chains of thought.

This raises an important question: will the ToT method, which also branches out from the step-by-step logic family, experience similar performance decreases when used with smaller parameter models? To address this question, my research aims to observe the performance of ToT prompting methods when applying LLMs of smaller parameters. We replicate three previous ToT experiments using open-sourced, smaller parameter models: Llama3-8b, which has 8 billion parameters, and Llama2-7b, which has 7 billion parameters. Our experiment focuses on three tasks, the Game of 24 is a 100 game of 24 problem sets using tree different prompting strategies (Few-shot, few-shot with CoT, and ToT) and the sudoku tasks, using three different sizes (3x3, 4x4, and 5x5) and four different prompting strategies (zero-shot, one-shot with CoT, few-shot with CoT, and ToT). Finally, we examine the natural language generation capability by introducing the creative writing task originally conducted in Yao's research. We use the open-source framework Ollama to execute the ToT code and run Llama2-7b and Llama3-8b respectively, locally on our machine.

Our experiment aims to provide a clear comparison of the ToT method's application to models with different parameters, building upon the previous research using GPT-4 and GPT-3.5-turbo. We expect the accuracy to decline when applied to smaller models, but we will also analyze the decay of standard prompting and CoT on smaller models to determine whether ToT experiences a similar accuracy decline, or ToT is resilient and performed strongly despite the cut in LLM parameters.

Our findings suggest that ToT may not be an effective method for improving the performance of long-range tasks with small parameter LLMs. The performance of Llama2-7b and Llama3-8b using ToT was the same or lower than using other prompting strategies in our two benchmark experiments conducted, and the Llama model struggled to generate accurate and coherent logic. These results are similar to the previous research on the performance decay of Chain-of-Thought in smaller parameter LLMs and that additional training data or modified prompting strategies may be needed to improve their performance.

Our study contributes to the understanding of the limitations of smaller parameter LLMs in long-range tasks and the ineffectiveness of the ToT method in improving their performance. The results of this study have important implications for the development of LLMs and their applications in natural language processing tasks. Future research can build upon our findings to explore the use of different prompts, different models, or other methods for improving the performance of LLMs in long-range tasks. Our findings will contribute to the understanding of the limitations and potential applications of ToT in long-range tasks, providing valuable insights for future research in this area.

2 Large language models (LLMs)

2.1 Understanding Large language models (LLMs)

Large Language Models (LLMs) are a class of artificial intelligence (AI) systems that have demonstrated remarkable capabilities in understanding, generating, and manipulating human language. These models are typically based on deep learning architectures, especially transformer models, and are trained on vast amounts of text data to perform a wide range of natural language processing (NLP) tasks. The development and capabilities of LLMs have profound implications for both academia and industry, revolutionizing how we interact with machines and process information.

2.1.1 Notation and Definitions

1. Pre-trained Language Model (LM):

We denote a pre-trained language model by p_θ, where θ represents the parameters of the model. These parameters are typically the weights and biases learned during the training process.

2. Language Sequence:

A language sequence is a sequence of tokens (words, characters, or subwords). We use lowercase letters like x, y, z, s, etc., to denote these sequences.

A specific language sequence x is written as $x = (x[1], x[2], \ldots, x[n])$, where $x[i]$ represents the i-th token in the sequence.

3. Probability of a Sequence:

The probability of a sequence x under the language model p_θ is given by the product of the conditional probabilities of each token in the sequence, given all previous tokens. Mathematically, this is expressed as:

$$p_\theta(x) = \prod_{i=1}^{n} p_\theta(\, x[i] \mid x[1], \ldots, x[i-1]\,)$$

This equation reflects the chain rule of probability, which states that the joint probability of a sequence can be decomposed into the product of conditional probabilities.

4. Collection of Language Sequences:

Uppercase letters (e.g., S) are used to denote a collection or set of language sequences.

The equation $(p_\theta(x) = \prod_{i=1}^{n} p_\theta(\, x[i] \mid x[1], \ldots, x[i-1]\,))$ expresses the probability of the entire sequence x as the product of the probabilities of each token $x[i]$ given the preceding tokens in the sequence.

For the LLM to work, it must follow the following procedure:

1. Token-by-Token Generation:

The model generates each token in the sequence one by one. For the first token $x[1]$, the probability is $p_\theta(x[1])$, since there are no preceding tokens.

2. Conditional Probability:

For the second token $x[2]$, the model generates it based on the first token $x[1]$, so the probability is $p_\theta(x[2] \mid x[1])$.

3. Chain Rule Application:

For the third token $x[3]$, the model considers both the first and second tokens, giving $(p_\theta(x[3] \mid x[1], x[2]))$. This process continues until the n-th token, where the probability is $(p_\theta(x[n] \mid x[1], \ldots, x[n-1]))$.

4. Combining Probabilities:

To get the total probability of the sequence x, we multiply all these conditional probabilities together:

$$p_\theta(x) = p_\theta(x[1]) \cdot p_\theta(x[2] \mid x[1]) \cdot p_\theta(x[3] \mid x[1], x[2]) \cdots p_\theta(x[n] \mid x[1], \ldots, x[n-1])$$

In essence, the equation captures the idea that the probability of a sequence can be computed by considering the contribution of each token, conditioned on the tokens that precede it. This is a fundamental concept in language modeling, where the goal is to model the likelihood of a sequence of tokens in a way that captures the dependencies between them.

2.1.2 Historical Context and Development

The foundation for LLMs can be traced back to earlier AI and NLP research. The transition from rule-based systems to statistical and machine-learning approaches marked a significant milestone in the evolution of NLP. Early models like Hidden Markov Models (HMMs) and Recurrent Neural Networks (RNNs) laid the groundwork, but they had limitations in handling long-range dependencies in text.

The introduction of the Transformer architecture by Vaswani et al. [1] was a pivotal moment. The transformer model's attention mechanism allowed it to weigh the importance of different words in a sentence, enabling better context understanding and overcoming the limitations of RNNs (Recurrent Neural Networks). This innovation led to the development of models such as BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer).

2.1.3 Technical Architecture

LLMs like GPT-3 (Generative Pre-trained Transformer 3) are based on the transformer architecture. They consist of multiple layers of attention mechanisms and feed-forward neural networks. Each layer processes the input data, refining its understanding through attention heads that focus on different parts of the input sequence.

The attention mechanism is central to the transformer's performance. It allows the model to assign different weights to different words in a sentence, capturing the nuances of language more effectively. This mechanism operates through self-attention, where each word in the input sequence is compared with every other word, and contextual embeddings are generated [1].

2.1.4 Training and Data

Training LLMs require substantial computational resources and vast datasets. These models are pre-trained on diverse text corpora, which include books, articles, websites, and other textual sources. The pre-training phase involves unsupervised learning, where the model learns to predict the next word in a sentence, thereby capturing the statistical properties of the language [3].

After pre-training, LLMs undergo fine-tuning on specific tasks with labeled data, which enhances their performance on those tasks. For instance, fine-tuning can be applied to tasks like machine translation, summarization, and question-answering [4].

2.1.5 Applications and Implications

LLMs have a wide array of applications across various domains. They excel in tasks such as:

1. Text Generation: LLMs can generate coherent and contextually relevant text, making them useful for content creation, chatbots, and creative writing.
2. Machine Translation: Models like BERT and GPT can translate text between languages with high accuracy.
3. Sentiment Analysis: LLMs can analyze the sentiment of text, which is valuable for market analysis, customer feedback, and social media monitoring.
4. Question Answering: These models can understand and respond to queries, aiding in customer support and information retrieval.
5. Summarization: LLMs can condense long texts into concise summaries, useful for news aggregation and document analysis [2].

2.1.6 Ethical Considerations and Challenges

Despite their capabilities, LLMs pose several ethical and practical challenges. One significant concern is bias. Since LLMs learn from large datasets that reflect societal biases, they can inadvertently propagate and amplify these biases in their outputs. Researchers have highlighted instances where LLMs produced biased or harmful content, raising questions about fairness and accountability in AI [5].

Another challenge is the environmental impact. Training large models requires significant computational power, leading to substantial energy consumption. Efforts are being made to develop more efficient training methods and to use renewable energy sources to mitigate this issue [6].

Moreover, the potential misuse of LLMs is a critical concern. They can be used to generate misleading or harmful content, necessitating robust mechanisms to detect and prevent misuse. This includes developing better AI governance frameworks and regulatory policies [7].

2.1.7 Future Directions

The future of LLMs holds exciting prospects and challenges. Ongoing research aims to make these models more efficient, interpretable, and fair. There is also a focus on multimodal models that integrate text with other data types like images and audio, broadening the scope of AI applications [8].

One promising direction is the development of models that require less data and computational resources, making advanced NLP accessible to a broader range of users and industries. Additionally, advancements in transfer learning and continual learning are expected to enhance the adaptability of LLMs to new tasks and languages with minimal retraining [9]

Overall, large Language Models represent a significant leap in artificial intelligence, transforming how machines understand and generate human language. While they offer immense potential across various fields, addressing their ethical, environmental, and practical challenges is crucial for their sustainable and responsible deployment. The ongoing advancements and research in this area promise to further expand the capabilities and applications of LLMs, making them an integral part of future technological developments.

2.2 Parameters of LLMs

The parameters of Large Language Models (LLMs) are critical to their performance and functionality. Parameters in the context of LLMs refer to the weights and biases within the neural network that are adjusted during training to minimize error in the model's predictions. The number of parameters is a key determinant of a model's capacity to learn and represent complex patterns in data. For example, large models like GPT-4, developed by OpenAI, has 1760 billion parameters, making it one of the largest LLMs to date. The sheer scale of these parameters allows the model to capture nuanced patterns in language and generate highly coherent and contextually relevant text.

2.2.1 Understanding Parameters

In LLMs, parameters are divided into two main types: weights and biases. Weights are the coefficients that are multiplied by input data, while biases are additional constants added to the transformations. Each neuron in the neural network has its own set of weights and biases, and the adjustment of these values during training enables the network to learn from the data.

The architecture of LLMs, particularly those based on the Transformer model, employs layers of multi-head self-attention and feed-forward networks. Each layer has its own parameters, and the self-attention mechanism involves calculating the relationships between different words in a sentence using these parameters. The complexity and depth of the model, indicated by the number of layers and heads, contribute to the total parameter count. In the case of GPT-3, its 96 layers and 96 attention heads per layer result in an enormous number of parameters, facilitating its ability to understand and generate diverse language patterns.

2.2.2 Training and Parameter Optimization

Training LLMs involves adjusting parameters through a process called backpropagation, which uses gradient descent to minimize a loss function. The loss function measures the difference between the model's predictions and the actual outcomes. During training, the model iterates over vast datasets, continuously updating its parameters to improve accuracy. This iterative process requires significant computational resources, as each parameter update involves complex calculations across potentially billions of parameters.

Pre-training and fine-tuning are two critical phases in training LLMs. Pre-training involves unsupervised learning on a large corpus of text, enabling the model to develop a broad understanding of language. Fine-tuning, on the other hand, involves supervised learning on task-specific data, refining the model's parameters for particular applications such as translation or summarization.

2.2.3 Impact of Parameter Scale

The scale of parameters in LLMs has a direct impact on their performance. Larger models with more parameters can capture more intricate details and subtle patterns in the data, leading to more accurate and coherent outputs. For instance, GPT-3's performance on various NLP benchmarks significantly surpasses that of its predecessors, largely due to its increased parameter count. However, this increase in parameters also brings challenges such as higher computational costs, longer training times, and increased energy consumption.

Moreover, the vast number of parameters makes these models more prone to overfitting, where the model learns the training data too well and performs poorly on new, unseen data. To mitigate this, techniques such as regularization, dropout, and early stopping are employed during training.

2.2.3 Future Directions

The trend in developing LLMs has been towards increasing the number of parameters to enhance performance. However, researchers are also exploring more efficient architectures and training techniques to achieve similar or better performance with fewer parameters. Techniques like sparse attention, model pruning, and quantization aim to reduce the computational burden without compromising the model's capabilities. Additionally, innovations in distributed training and hardware acceleration are helping to manage the immense computational requirements of training large models.

Overall, the parameters of Large Language Models are fundamental to their ability to understand and generate human language. While the massive scale of parameters in models like GPT-3 enables advanced language processing capabilities, it also presents significant challenges in terms of computational resources and efficiency. Ongoing research aims to balance these factors, ensuring that future LLMs continue to advance in both performance and accessibility.

2.3 Temperature in LLMs

Temperature in the context of large language models (LLMs) refers to a parameter used in the softmax function to control the randomness of predictions. It is a crucial hyperparameter that influences the diversity and creativity of the generated text. Adjusting the temperature can significantly affect the model's output by modifying the probability distribution of the next word prediction.

2.3.1 Explanation of Temperature

In a typical language model, the output layer consists of a probability distribution over the vocabulary. The softmax function is applied to the logits (the raw, unnormalized scores output by the model) to convert them into probabilities. The temperature parameter T modifies these logits before the softmax function is applied, as follows:

$$P_i = \frac{\exp\left(\frac{\backslash logit_i}{T}\right)}{\sum_j \exp\left(\frac{\backslash logit_j}{T}\right)}$$

Where:

- $\backslash logit_i$ is the logit for the (i)-th word.

- T is the temperature.

- Pi is the resulting probability after applying softmax.

When $T = 1$, the logits remain unchanged. When $T > 1$, the probability distribution becomes more uniform (higher entropy), making the model more random. Conversely, when $T < 1$, the distribution becomes sharper (lower entropy), making the model's predictions more deterministic and confident.

2.3.2 Academic Research on Temperature

Several studies have explored the impact of temperature on language models. Some key findings include:

1. Impact on Creativity and Coherence:

Radford et al. [3] in the paper introducing GPT-2 discuss how varying temperature can balance between generating more creative and diverse text versus more predictable and repetitive text. Higher temperatures lead to more creative outputs but can result in less coherent sentences, while lower temperatures produce more predictable and coherent text .

2. Role in Controlling Output Quality:

Holtzman et al. [10] in their study on "The Curious Case of Neural Text Degeneration" highlight how temperature affects the generation of high-quality text. They suggest that lower temperatures (close to zero) can help in avoiding degenerate text sequences where the model repetitively outputs the same phrase .

3. Balancing Diversity and Relevance:

Ziegler et al. [11] in their work on fine-tuning language models for human preferences discuss how temperature tuning is essential for balancing the diversity and relevance of generated content. They emphasize that appropriate temperature settings can align model outputs more closely with human preferences, providing a better balance between randomness and relevance .

2.3.3 Practical Applications

In practice, the choice of temperature depends on the specific application and desired outcome:

For tasks requiring highly creative and varied text (e.g., storytelling, poetry), higher temperatures are preferable.

For applications needing precise and reliable information (e.g., technical documentation, customer support), lower temperatures are beneficial.

To conclude, temperature is a vital hyperparameter in LLMs that significantly influences the model's behavior by controlling the randomness of its output. Understanding and tuning the temperature parameter allows for better control over the generated text, balancing between creativity and coherence based on the specific requirements of the application.

2.4 GPT-3.5-turbo

GPT-3.5-turbo is an advanced language model developed by OpenAI, based on the GPT-3 architecture with enhancements and optimizations to improve performance and efficiency. The model has only 20 billion parameter size. This model belongs to the family of Generative Pre-trained Transformers (GPT), which leverage a transformer architecture for natural language processing tasks.

2.4.1 Key Features and Improvements

1. Architecture and Scale:

GPT-3.5-turbo maintains the core transformer architecture introduced by Vaswani et al. [1], which uses self-attention mechanisms to process input sequences. Like its predecessor GPT-3, GPT-3.5-turbo likely features an extensive number of parameters, running into hundreds of billions, which allows it to perform a wide array of language tasks with remarkable accuracy and coherence [1][4].

2. Training Data and Process:

The model has been trained on a diverse and large corpus of text data sourced from the internet.

This includes a mixture of licensed data, data created by human trainers, and publicly available data. This extensive training dataset enables GPT-3.5-turbo to generate human-like text and understand context in a nuanced manner [4].

3. Optimization Techniques:

GPT-3.5-turbo incorporates several optimization techniques over GPT-3. These include improvements in the training process, such as better handling of long-range dependencies in text, enhancements in the model's ability to follow instructions, and fine-tuning on specific datasets to improve performance on particular tasks [1][3].

4. Performance and Capabilities:

The turbo variant of GPT-3.5 is designed to be faster and more cost-effective, achieving lower latency in generating responses while maintaining high-quality output. This makes it particularly suitable for real-time applications and interactive use cases where response speed is crucial [4].

5. Applications:

GPT-3.5-turbo is versatile and can be applied across various domains such as:

Text Generation: Creating content, writing stories, or drafting emails.

Question Answering: Providing answers to queries based on provided context or general knowledge.

Translation: Translating text between different languages.

Summarization: Summarizing long documents into concise versions.

Conversational Agents: Powering chatbots and virtual assistants to handle customer queries and support tasks [3][4].

In general, GPT-3.5-turbo represents a state-of-the-art advancement in natural language processing, building on the strengths of its predecessors while introducing critical improvements in performance and efficiency. Its wide range of applications and the ongoing efforts to ensure ethical use make it a powerful tool in the field of AI.

2.5 GPT-4

GPT-4, developed by OpenAI, represents a significant advancement in the realm of artificial intelligence, specifically within large language models (LLMs). This multimodal model is capable of processing both image and text inputs to generate text outputs, marking a notable improvement over its predecessors like GPT-3.5. The development and implementation of GPT-4 highlight several critical aspects, including its performance, capabilities, and limitations, as outlined in the GPT-4 Technical Report [12].

2.5.1 Performance and Capabilities

GPT-4 excels in a wide array of professional and academic benchmarks. For instance, it performs remarkably well in simulated bar exams, achieving scores in the top 10% of test takers, a stark contrast to GPT-3.5, which ranked in the bottom 10% [12]. Additionally, GPT-4 outperforms existing models in traditional natural language processing (NLP) benchmarks and surpasses state-of-the-art systems in multiple tasks without the need for task-specific fine-tuning [12].

A standout feature of GPT-4 is its performance on the MMLU (Massive Multitask Language Understanding) benchmark. It not only excels in English but also shows strong performance in 24 out of 26 other languages tested, outperforming the English-language state-of-the-art models in many cases. This multilingual capability underscores GPT-4's versatility and its potential for global applications.

2.5.2 Safety and Alignment Improvements

One of the significant improvements in GPT-4 over its predecessors is its enhanced alignment with factual accuracy and desired behavior. The model underwent rigorous post-training alignment processes, which resulted in a notable reduction of hallucinations and reasoning errors. For instance, GPT-4 demonstrates a 19 percentage point improvement in factual accuracy over GPT-3.5 based on internal adversarial evaluations [12].

Moreover, GPT-4 has shown significant progress in handling disallowed content and sensitive requests. Compared to GPT-3.5, GPT-4 is 82% less likely to respond to disallowed prompts and adheres to policy guidelines 29% more frequently when dealing with sensitive information. These improvements are critical for deploying AI in real-world applications where safety and reliability are paramount.

2.5.3 Limitations and Future Directions

Despite its advancements, GPT-4 is not without limitations. It still experiences hallucinations, albeit to a lesser extent than previous models, and can make reasoning errors. Furthermore, GPT-4's knowledge is primarily based on pre-training data up to September 2021, which limits its awareness of more recent events [12].

The model also faces challenges with certain complex reasoning tasks and can sometimes be overly gullible, accepting false statements from users. Additionally, GPT-4, like its predecessors, has a limited context window and does not learn from its interactions, which restricts its ability to adapt and improve over time.

In conclusion, GPT-4 represents a significant leap forward in the development of large language models. Its enhanced performance, multilingual capabilities, improved safety measures, and multimodal functionalities make it a versatile and powerful tool in the field of AI. However, ongoing research and development are necessary to address its limitations and fully realize its potential in various high-stakes applications [12].

2.6 Llama2

Llama 2, developed by Meta, is also a significant advancement in the field of large language models

(LLMs). This model family includes both pretrained and fine-tuned versions, specifically optimized for dialogue use cases. Llama 2 models range from 7 billion to 70 billion parameters, offering a robust solution for various AI applications. The Llama 2-Chat variants, which are fine-tuned, outperform many existing open-source chat models on multiple benchmarks and are competitive with some closed-source models in terms of helpfulness and safety based on human evaluations.

2.6.1 Pretraining Methodology

The pretraining of Llama 2 builds upon the techniques used for its predecessor, Llama 1, with several enhancements aimed at improving performance and scalability. Notably, Llama 2 models were trained on 2 trillion tokens of data, sourced from a new mix of publicly available datasets, excluding data from Meta's products or services. The training data was carefully curated to remove content from sites known to contain high volumes of personal information. This extensive and diverse dataset aims to enhance the factual accuracy of the models while minimizing hallucinations.

Llama 2 incorporates advanced architectural features such as grouped-query attention (GQA), which significantly improves the scalability of inference for larger models. The models were trained using the AdamW optimizer with specific hyperparameters designed to optimize learning rates and decay schedules. [13]

2.6.2 Fine-Tuning Approach

The fine-tuning process for Llama 2, particularly the Llama 2-Chat models, involves a multi-stage process to align the models with human preferences and enhance their usability in dialogue contexts.

The initial fine-tuning is performed through supervised learning, where the models are trained on annotated datasets. This is followed by reinforcement learning with human feedback (RLHF), employing techniques like rejection sampling and Proximal Policy Optimization (PPO). These methodologies ensure that the models iteratively improve based on human feedback, refining their responses to be more aligned with user expectations and safety standards. [13]

2.6.3 Performance and Evaluation

Llama 2-Chat models have demonstrated superior performance on a variety of benchmarks compared to other open-source models. Human evaluations for helpfulness and safety have shown that these models are generally on par with some closed-source alternatives. The evaluation process involves both single and multi-turn prompts to comprehensively assess the models' capabilities in interactive settings.

The detailed pretraining and fine-tuning methodologies, combined with rigorous safety protocols, make Llama 2 a valuable resource for the AI community. The open release of these models, accompanied by a responsible use guide and code examples, aims to foster further research and development in AI, encouraging the community to build on these advancements and contribute to the responsible deployment of LLMs.

2.7 Llama 3

In early 2024, Meta has introduced Llama 3, the latest generation of their large language model

(LLM), marking a significant leap in the capabilities and performance of open-source LLMs. This new version of Llama includes enhanced safety tools, new capabilities, and improved performance over its predecessor, Llama2.

2.7.1. Model Architecture and Efficiency

Llama 3 maintains a standard decoder-only transformer architecture but introduces several critical improvements. The tokenizer in Llama 3 has a vocabulary of 128K tokens, significantly enhancing language encoding efficiency. Grouped Query Attention (GQA) is utilized across all model sizes, improving inference efficiency. [17]

2.7.2. Training Data

Llama 3's training involved over 15 trillion tokens, seven times more than Llama 2, sourced from publicly available data. The dataset includes a substantial portion of non-English data, preparing the model for multilingual applications. The training data underwent rigorous quality filtering, using heuristic filters, NSFW filters, and semantic deduplication. [17]

2.7.3 Scaling and Pretraining

Llama 3's training utilized extensive compute resources and advanced parallelization techniques. The models continued to improve with increasing amounts of training data, even beyond the Chinchilla-optimal amount. The efficiency improvements in training infrastructure resulted in a threefold increase in training efficiency compared to Llama2. [17]

2.7.4. Instruction Fine-Tuning

The fine-tuning process for Llama 3 incorporated supervised fine-tuning (SFT), rejection sampling, Proximal Policy Optimization (PPO), and Direct Preference Optimization (DPO). This approach enhanced the models' performance on reasoning and coding tasks by leveraging preference rankings from human evaluations.

2.7.5. Performance and Evaluation

Llama 3 models, particularly the 8B and 70B parameter versions, exhibit state-of-the-art performance on various industry benchmarks. They have shown significant improvements in reasoning, code generation, and instruction following, with lower false refusal rates and increased response diversity.

2.7.6 Differences Between Llama 2 and Llama 3

1.Parameter Scale: Llama 3 introduces models with up to 70 billion parameters, and future releases will include models exceeding 400 billion parameters, whereas Llama 2's models max out at 70 billion parameters.

2. Tokenizer and Inference Efficiency: The tokenizer in Llama 3 has a larger vocabulary (128K tokens) and is more efficient, yielding up to 15% fewer tokens compared to Llama 2.

3.Training Data Volume: Llama 3's training dataset is seven times larger than that of Llama 2, including a significant increase in code-related data and non-English content.

4.Performance Improvements: Llama 3 shows substantial improvements in reasoning, code generation, and instruction following. The training efficiency and model performance at scale are also markedly enhanced.

5.Safety Tools: Llama 3 integrates advanced safety mechanisms like Llama Guard 2 and Code Shield, enhancing security and responsible use compared to Llama 2.

Overall, Llama 3 sets a new benchmark for open-source LLMs with its advanced architecture, extensive training, and robust safety features. Meta's commitment to an open-source ethos aims to drive innovation and responsible AI development. The improvements from Llama 2 to Llama 3 demonstrate Meta's dedication to advancing AI capabilities while ensuring safety and usability for a broad range of applications.

3. Prompt Engineering

3.1 Introduction to Prompt Engineering

Prompt engineering is an emerging field in artificial intelligence that focuses on designing and refining input prompts to effectively guide language models in generating desired outputs. As large language models become increasingly sophisticated, the art of crafting prompts has become crucial for optimizing their performance across various tasks. Prompt engineering encompasses techniques such as few-shot learning, chain-of-thought prompting, and instruction tuning to enhance model capabilities without the need for fine-tuning. Wei et al. [15] demonstrate that carefully constructed prompts can significantly improve a model's ability to perform complex reasoning tasks. Moreover, Kojima et al [14]. highlight the potential of zero-shot prompting in enabling models to tackle novel problems without task-specific training data. As the field evolves, researchers continue to explore innovative prompt design strategies to unlock the full potential of language models in diverse applications.

In this chapter, we will discuss every prompting method in detail.

3.2 Input-Output (IO) prompting

3.2.1 Notation and Definitions

Input-output (IO) prompting is a common technique used to transform a given problem input x into a desired output y using a language model (LM). This is done by designing a specific prompt, called $(\mathrm{promptIO}(x))$, which encapsulates the input (x) along with task instructions and/or a few examples of input-output pairs. The language model then predicts the output (y) based on this prompt. Mathematically, this can be expressed as:

$$[y \sim p_\theta(y \mid \mathrm{promptIO}(x))]$$

This equation means that the output (y) is drawn from a probability distribution (p_θ) given the prompt $(\mathrm{promptIO}(x))$, where (θ) represents the parameters of the language model.

For simplicity, we can denote this process as (p_{prompt}):

$$[p_{\text{prompt}} = p_\theta(\text{ output } | \text{ input }) = p_\theta\big(\text{ output } \big| \text{ prompt(input) }\big)]$$

Here, (prompt(input)) is a function that transforms the input into the prompt format, including instructions or examples necessary for the model to understand and solve the task.

Thus, IO prompting can be formulated more succinctly as:

$$y \sim p_{\text{IO}}\theta(y \mid x)$$

This notation emphasizes that the language model is generating the output $(\, y\,)$ based on the input $(\, x\,)$ after it has been processed into a prompt by $(\text{promptIO}(x))$.

3.2.2 Zero-shot prompting

Zero-shot prompting enables large language models (LLMs) to perform tasks without any prior examples or explicit task-specific tuning. The core concept involves conditioning the model to generate responses based on minimal instructions or a simple prompt. For instance, adding a prompt like "Let's think step by step" before a question can significantly enhance the model's reasoning capability. This approach was demonstrated in the Zero-shot-CoT (Chain of Thought) method, where the prompt encourages the model to break down a problem into logical steps, leading to a correct and coherent answer. The Zero-shot-CoT approach outperformed standard zero-shot prompting in various reasoning tasks, including arithmetic and symbolic reasoning, by generating intermediate steps in the reasoning process. [14] Unlike few-shot prompting, which requires carefully crafted examples, zero-shot prompting leverages the model's inherent capabilities with minimal prompt engineering, making it a versatile and powerful tool for diverse tasks.

An example prompt for zero-shot prompting will be:

Input: "Read carefully and classify the following text as either positive, negative, or neutral sentiment: "

Output: 'The new restaurant down the street has terrible service and mediocre food.'

In this example, the model is asked to classify the sentiment of a given text without being provided any labeled examples of positive, negative, or neutral sentiments. The prompt clearly states the task (sentiment classification) and the possible categories (positive, negative, neutral) but doesn't give any training data. The model must use its pre-existing knowledge to understand the concept of sentiment and apply it to the given text.

3.2.3 Few-shot prompting

Few-shot prompting enables large language models (LLMs) to perform tasks using only a handful of examples or simple instructions, akin to how humans can often learn new tasks quickly from minimal data. This technique was exemplified in the development of GPT-3, a 175 billion-parameter autoregressive language model, which demonstrated substantial improvements in few-shot performance across various natural language processing (NLP) tasks. Unlike traditional models that require extensive fine-tuning on large, task-specific datasets, GPT-3 can rapidly adapt to new tasks with just a few examples provided in the input prompt. This approach allows GPT-3 to achieve competitive results on tasks like translation, question-answering, and reading comprehension, sometimes even surpassing state-of-the-art models that rely on extensive fine-tuning. The model's ability to generalize from minimal context showcases its versatility and potential for practical applications in diverse scenarios, reducing the dependency on large annotated datasets and enabling more flexible and efficient task handling. [4]

An example for a few-shot prompt will be:

Input: "Classify the sentiment of the following reviews as positive, negative, or neutral:

1. 'This movie was amazing!' - Positive
2. 'I didn't like the book at all.' - Negative
3. 'The weather is okay today.' - Neutral

Now classify this review:

'The new smartphone has great features but it's quite expensive.'"

In this example, the model is presented with three labeled examples of sentiment classification before being asked to classify a new, unlabeled review. The few examples serve as a mini-training set, helping the model understand the task and calibrate its responses.

3.3 Chain-of Thought (CoT) prompting

Chain of thought prompting is another way to guide large language models through the reasoning process. By showing us examples of step-by-step thinking, it encourages large language models to follow a similar pattern when solving problems or answering questions. This leads to more accurate and transparent answers, especially for complex tasks.

3.3.1 Notation and Definitions

Chain-of-thought (CoT) prompting is a method developed to tackle complex problems where the direct mapping of an input (x) to an output (y) is challenging. This is particularly useful in scenarios such as solving math questions where (x) is the problem statement and (y) is the final numerical answer.

The core idea behind CoT prompting is to introduce a series of intermediate steps, denoted as $(z_1, z_2, \ldots, z_n)$, to bridge the gap between the input (x) and the output (y). Each (z_i) represents a coherent language sequence that serves as a meaningful intermediate step in the problem-solving process. For example, in a math question, (z_i) could be an intermediate equation or a reasoning step.

To solve problems using CoT prompting, the sequence of thoughts (z_i) is generated one step at a time, starting from (x) and considering all previous steps. Mathematically, this is expressed as:

$$z_i \sim p_{\mathrm{CoT}}^{\theta}(z_i \mid x, z_1, \ldots, z_{i-1})$$

This means that each intermediate step (z_i) is sampled from a probability distribution $(p_{\mathrm{CoT}}^{\theta})$, conditioned on the input (x) and all previously generated steps $(z_1) to (z_{i-1})$.

Once all intermediate steps $(z_1, \ldots, z_n)$ have been generated, the final output (y) is then sampled from the distribution:

$$y \sim p_{\mathrm{CoT}}^{\theta}(y \mid x, z_1, \ldots, z_n)$$

In practical applications, the entire sequence $([z_1, \ldots, z_n, y])$ is often sampled as a continuous language sequence. This means the model generates the intermediate thoughts and the final output together in one go. The exact decomposition of these thoughts—whether each z_i is a phrase, a sentence, or a paragraph—is typically left unspecified, allowing the model to decide the best way to structure the intermediate steps.

3.3.2 Description and experiment results

Wei et al [15] were first to explore the chain-of-thought prompting, which significantly enhances the reasoning capabilities of large language models (LLMs). The core idea is to provide intermediate reasoning steps in natural language during the prompting process, allowing models to perform better on complex tasks such as arithmetic, commonsense, and symbolic reasoning.

Traditional prompting methods, which only offer input-output pairs, often fall short in tasks requiring deeper reasoning. Chain-of-thought prompting, however, involves giving examples that include a series of intermediate steps leading to the final answer. This approach mirrors human problem-solving processes, where breaking down problems into smaller steps often makes them easier to solve. By incorporating these intermediate steps, models can better understand and solve complex tasks.

The authors conducted extensive experiments on three large language models, including the Pathways Language Model (PaLM) with 540 billion parameters. They evaluated the effectiveness of chain-of-thought prompting using various benchmarks. For arithmetic reasoning, models were tested on datasets like GSM8K, SVAMP, and ASDiv. [15] The results showed that chain-of-thought prompting dramatically improved accuracy compared to standard prompting. Similarly, for commonsense reasoning tasks, such as those found in the AQuA dataset, and symbolic reasoning tasks, this method again led to superior performance.

An ablation study further highlighted the importance of generating intermediate steps, demonstrating that these steps are crucial for the improved performance observed. The robustness of chain-of-thought prompting was confirmed across different tasks and datasets, indicating that this method consistently enhances the reasoning capabilities of LLMs.

In conclusion, chain-of-thought prompting is a simple yet powerful technique that unlocks the reasoning potential of large language models. By providing intermediate reasoning steps, models can tackle complex tasks more effectively, leading to significant performance gains. This method has broad applicability and can potentially revolutionize how LLMs are used for various reasoning-intensive tasks. The paper underscores the importance of intermediate steps in problem-solving and opens new avenues for improving AI's reasoning abilities.

3.3.3 Parameter limitations in Chain-of-Thought

It was worth noting that, in his discovery, wei et al [15] demonstrates the impact of chain-of-thought (CoT) prompting on models of various sizes. CoT prompting enhances the reasoning abilities of language models by including intermediate steps in the prompts, enabling models to solve complex tasks more effectively. The performance improvements are especially pronounced with larger models, revealing that CoT is an emergent ability that significantly benefits from increased model scale.

The paper evaluates the performance of different models on arithmetic reasoning benchmarks such as GSM8K, SVAMP, ASDiv, AQuA, and MAWPS. Here's a breakdown of the results:

UL2 20B: The performance gain from CoT prompting is minimal, indicating that smaller models (e.g., 20B parameters) do not benefit substantially from CoT prompting.

LaMDA (420M - 137B): Performance improves noticeably as the model size increases. For example, LaMDA 137B shows significant gains across multiple benchmarks with CoT prompting, with accuracy improvements ranging from 6.5% to 14.7% on tasks like GSM8K and MAWPS.

GPT-3 (350M - 175B): The 175B model shows dramatic improvements with CoT prompting, with performance gains of over 30% on GSM8K and notable improvements on other benchmarks as well. Smaller versions of GPT-3, however, do not show substantial benefits.

PaLM (8B - 540B): The 540B model exhibits substantial gains from CoT prompting, with improvements ranging from 1.8% to 39.0% across various benchmarks. This indicates that the effectiveness of CoT prompting scales positively with model size.

Meanwhile, for commonsense reasoning tasks, similar trends are observed:

UL2 20B: The gains from CoT prompting are modest, suggesting limited benefits for smaller models.

LaMDA and GPT-3: Both show significant improvements with larger models. For instance, GPT-3 175B and LaMDA 137B exhibit improvements of up to 31.3% on tasks like CSQA and StrategyQA.

PaLM 540B: Again, significant performance improvements are observed, with gains of up to 14.2% on various commonsense reasoning benchmarks. [15]

On symbolic manipulation task benchmarks, including last letter concatenation and coin flip state tracking, the experiment observes that:

UL2 20B and Smaller Models: Minimal gains are observed, indicating that CoT prompting is less effective for smaller models.

Larger Models (68B and Above): Significant performance improvements are seen with larger models. For example, PaLM 540B shows dramatic gains in tasks like last letter concatenation, highlighting the importance of model scale for CoT prompting.

Overall, chain-of-thought prompting emerges as a powerful technique for enhancing the reasoning capabilities of large language models. However, its effectiveness is strongly correlated with model size. Smaller models (below 20B parameters) derive limited benefits, while larger models (175B parameters and above) show substantial performance gains. [15] This underscores the importance of model scale in leveraging the full potential of chain-of-thought prompting for complex reasoning tasks.

4. Tree-of-Thought Prompting Technique

4.1 Understanding Prompt Chaining in Large Language Models (LLMs)

Prompt chaining is an advanced technique used in the context of Large Language Models (LLMs) to enhance their performance on complex tasks by sequentially combining multiple prompts. This method leverages the ability of LLMs to generate contextually relevant responses, building on each successive prompt to achieve a more refined or comprehensive output. Prompt chaining has significant implications for improving the capabilities of LLMs in various applications, from natural language understanding to interactive dialogue systems.

4.1.1 Concept and Mechanism

Prompt chaining involves the sequential use of multiple prompts, where the output of one prompt serves as the input or context for the next. This chaining process allows the model to refine its responses iteratively, incorporating additional information or constraints provided by subsequent prompts. The technique exploits the transformer model's capacity to maintain context across long sequences, enabling the LLM to generate more coherent and contextually relevant text over multiple steps.

For example, in a task requiring complex reasoning or multi-step problem-solving, a single prompt might not suffice to generate a complete and accurate response. By using prompt chaining, the initial prompt can elicit a preliminary response, which can then be elaborated upon or refined through additional prompts. This approach can be particularly useful in tasks such as text summarization, question-answering, and dialogue generation, where maintaining and evolving context is crucial.

4.1.2 Technical Implementation

Implementing prompt chaining involves designing a sequence of prompts that guide the LLM through a multi-step reasoning process. Each prompt in the chain is crafted to build upon the previous responses, ensuring continuity and coherence. This can be achieved through manual prompt engineering, where domain experts create specific prompts tailored to the task, or through automated methods that dynamically generate and adjust prompts based on the model's responses.

Manual Prompt Engineering: Experts design a series of prompts that progressively narrow down the scope or detail of the task. For instance, in a text summarization task, the initial prompt might ask for a general summary, while subsequent prompts request additional details or focus on specific aspects of the text.

Automated Prompt Generation: Using algorithms to dynamically generate and adapt prompts based on the model's interim outputs can enhance the efficiency of the chaining process. Techniques such as reinforcement learning or evolutionary algorithms can be employed to optimize the sequence of prompts for better performance.

4.1.3 Applications and Benefits

Prompt chaining is particularly beneficial in scenarios where tasks are too complex to be addressed in a single step. Some notable applications include:

Complex Question-Answering: For intricate queries requiring multi-step reasoning, prompt chaining can break down the problem into smaller, manageable parts, leading to more accurate and detailed answers.

Text Summarization: By iteratively refining summaries, prompt chaining helps generate concise yet comprehensive summaries of lengthy documents.

Dialogue Systems: In conversational AI, prompt chaining enables more natural and coherent dialogues by maintaining context over multiple interactions, improving the overall user experience. Creative Writing: For tasks such as story generation, prompt chaining can enhance creativity and coherence by building on each successive narrative element provided by the model.

4.1.4 Challenges and Considerations

Despite its advantages, prompt chaining presents several challenges. Crafting effective prompt sequences requires a deep understanding of both the task and the model's behavior. Poorly designed prompts can lead to context drift, where the model's responses become less relevant or coherent over successive steps. Additionally, the increased computational overhead associated with generating and processing multiple prompts can impact efficiency and scalability. [16]

Another consideration is the potential for cumulative errors. Mistakes in earlier prompts can propagate through the chain, compounding inaccuracies and affecting the final output. Addressing these issues involves careful prompt design, iterative testing, and potentially incorporating error-correction mechanisms within the prompt chain. [16]

4.1.5 Future Directions

The future of prompt chaining in LLMs looks promising, with ongoing research focused on automating and optimizing the prompt generation process. Integrating advanced techniques such as reinforcement learning and fine-tuning models specifically for multi-step tasks can enhance the effectiveness of prompt chaining. Additionally, combining prompt chaining with other methodologies like few-shot learning and transfer learning can further improve the adaptability and performance of LLMs across diverse applications.

In summary, prompt chaining represents a powerful technique for leveraging the capabilities of Large Language Models in complex tasks requiring multi-step reasoning and context maintenance. By sequentially combining multiple prompts, this approach enhances the model’s ability to generate coherent and contextually relevant outputs. While there are challenges to address, the potential benefits make prompt chaining a valuable tool in advancing the capabilities of LLMs and expanding their applications in various fields.

4.2 Tree-of-Thought

The Tree of Thought (ToT) framework is an innovative approach designed to enhance the problem-solving abilities of large language models (LLMs), such as GPT-4. This method draws inspiration from the cognitive processes of the human mind, particularly the way humans tackle complex reasoning tasks through trial and error, exploring multiple potential solutions in a tree-like manner. Unlike traditional auto-regressive models, which generate responses in a linear, sequential fashion, ToT allows for backtracking and exploring alternative paths, thereby significantly improving the model's ability to handle intricate problems.

4.2.1 Notations and Components

1. State (s): A state in the ToT framework is represented as $(s = [x, z_1, \ldots, z_i])$, where $(\ x\ )$ is the

input and $(z_1, \ldots, z_i)$ are the sequence of thoughts generated so far.

2. Thought Generator $(G(p_\theta, s, k))$: This is a function that generates (k) candidate thoughts from the current state (s). The thought generator can use different strategies:

i.i.d. Sampling: Thoughts are sampled independently and identically distributed (i.i.d.) from a Chain-of-Thought (CoT) prompt, suitable for rich thought spaces (e.g., generating paragraphs).

Sequential Proposing: Thoughts are proposed sequentially using a propose prompt, suitable for more constrained thought spaces (e.g., generating words or lines).

3. State Evaluator $(V(p_\theta, S))$: This function evaluates the progress of different states towards solving the problem. There are two strategies for evaluation:

Independent Valuation: Each state is valued independently based on its prospects, generating a scalar value or classification.

-Voting Mechanism: States are compared, and a "good" state is voted out based on a comparison of different states in (S).

4. Search Algorithm: To navigate the tree of thoughts, different search algorithms can be used:

Breadth-First Search (BFS): Explores all nodes at the present depth level before moving on to nodes at the next depth level. Maintains a set of the (b) most promising states per step.

Algorithm 1: ToT-BFS

1. Initialization: Start with the initial state$(S_0 = \{x\})$.

2. Iteration (for each step (t) up to limit (T)):

Generate (k) candidates for the next thought step for each state $(s \in S_{t-1})$ using $(G(p_\theta, s, k))$.

Evaluate the generated states (S_t') using $(V(p_\theta, S_t'))$.

Select the top (b) states based on their evaluations to form (S_t).

3. Final Selection: Return the output generated from the best state in the final set (S_T).

Depth-First Search (DFS): Explores as far as possible along each branch before backtracking. Prunes subtrees from states deemed unpromising based on a value threshold(v_{th}).

Algorithm 2: ToT-DFS

1. Initialization: Start with the current state (s) and step (t).

2. Recursion $(while(t \leq T))$:

Generate (k) candidates for the next thought step for the current state (s) using $(G(p_\theta, s, k))$.

Sort and evaluate the generated states.

Recursively explore the most promising state (s') if its evaluation exceeds the threshold (v_{th}).

3. Backtracking: If the state is deemed unpromising, prune its subtree and backtrack to explore other candidates.

4.2.3 Understanding Tree-of-Thought

The core idea behind ToT is to emulate the human problem-solving process, which often involves branching out into different directions, reconsidering previous steps, and making adjustments as new information is gathered. This is akin to traversing a tree, where each node represents a potential solution or a partial step towards the solution, and branches represent different paths that can be taken from each node. The ToT framework augments the LLM with several additional modules: a prompter agent, a checker module, a memory module, and a ToT controller. These modules work together in a multi-round conversational process with the LLM to explore the solution space more effectively.

The prompter agent is responsible for initiating the problem-solving process by presenting the LLM with a problem description and encouraging it to generate intermediate solutions rather than attempting to solve the problem in a single step. Once an intermediate solution is generated, the checker module verifies its correctness. If the solution is valid, it is stored in the memory module, which keeps track of the entire conversation and state history. This allows the system to backtrack and explore other directions if necessary, coordinated by the ToT controller. The controller oversees the entire process, deciding when to continue exploring a current path and when to backtrack and try a different approach.

The ToT framework's effectiveness is highlighted through various experiments. For instance, in the context of solving Sudoku puzzles, the ToT-based solver showed a significant improvement in success rates compared to traditional methods. [19] The framework's modular structure allows it to be applied to a wide range of mathematical and logical reasoning tasks, from theorem proving to complex puzzles, showcasing its versatility and potential for generalization. [18]

4.2.4 Research Insights

Both Long [19] and Yao et al [18] discovered Tree-of-Thought prompting method independently in their research. Long presents a comprehensive introduction to the ToT framework. His research highlights the limitations of current auto-regressive LLMs in handling long-range reasoning tasks, which require a series of logical steps and planning. These models, despite their prowess in short-range reasoning, often falter when it comes to tasks that necessitate extended logical derivations and error corrections. Move on, his research discusses how the ToT framework addresses these limitations by incorporating a tree-like thought process. This process allows the system to backtrack and explore different paths, thus mitigating the issue of linear response generation. The introduction of modules like the checker for correctness verification and the memory module for state tracking ensures that the model can recover from errors and avoid dead-ends. The experimental results with a Sudoku solver demonstrate the framework's practical benefits, significantly increasing the success rate of solving such puzzles.

On the other hand, Yao’s study delves deeper into the technical implementation and the broader implications of the ToT framework. It emphasizes the deliberate and structured decision-making process enabled by ToT. It contrasts this with traditional Chain of Thought (CoT) prompting, which, while effective in eliciting step-by-step solutions, often lacks the ability to correct mistakes and explore alternative solutions comprehensively. In the paper, the Yao and his team propose a systematic approach where the LLM engages in a multi-round conversation guided by the ToT framework. This involves generating potential solutions, evaluating their correctness, and backtracking when necessary. The paper introduces a more detailed architecture for the ToT system, including specific strategies for the prompter agent, checker module, memory module, and ToT controller. These components work in concert to enhance the LLM's problem-solving capabilities, making the process more robust and reliable.

Yao’s paper also presents a series of experiments to validate the effectiveness of the ToT framework. [18] One notable experiment involves the Game of 24, where the ToT framework achieved a success rate of 74.3%, significantly outperforming CoT prompting and other traditional methods. This demonstrates the framework's ability to handle complex reasoning tasks more effectively, leveraging the structured exploration and backtracking capabilities inherent in the ToT approach.

Overall, the Tree of Thought framework represents a significant advancement in the field of artificial intelligence, particularly in enhancing the problem-solving capabilities of large language models. By emulating the human cognitive process of exploring multiple potential solutions and allowing for backtracking, ToT addresses the limitations of traditional auto-regressive models in handling long-range reasoning tasks. The research presented in the two papers underscores the framework's effectiveness and versatility, demonstrating its potential to tackle a wide range of complex problems in various domains.

In summary, ToT transforms the problem-solving paradigm for LLMs by integrating a tree-like exploration strategy, systematic decision-making, and robust error correction mechanisms. This framework not only improves the accuracy and reliability of solutions but also expands the range of tasks that LLMs can effectively handle, paving the way for more sophisticated and capable AI systems in the future.

4.3 Comparing Chain-of-Thought with Tree-of-Thought

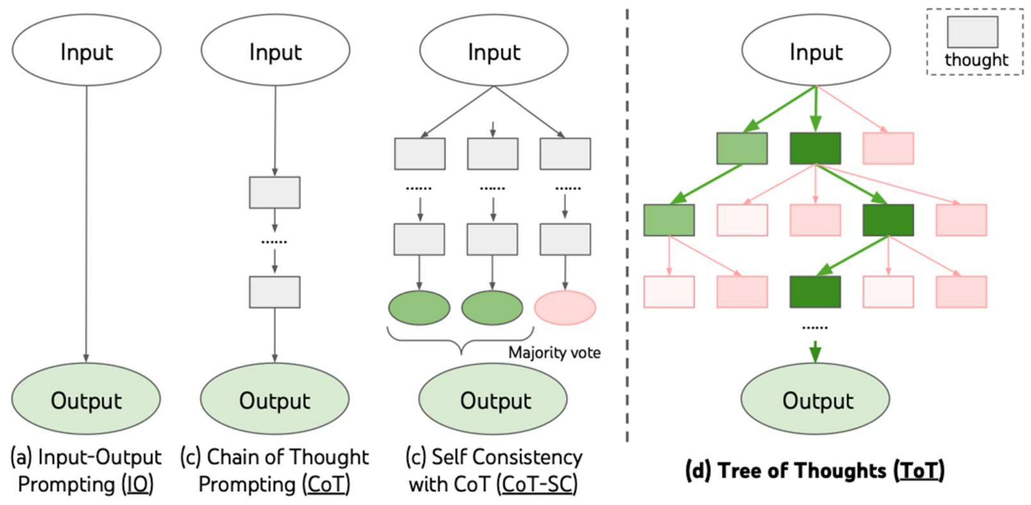
Input
Input
Input
Input
thought
Majority vote
Output
Output
Output
Output
(a) Input-Output Prompting (IO)
(c) Chain of Thought Prompting (CoT)
(c) Self Consistency with CoT (CoT-SC)
(d) Tree of Thoughts (ToT)

Fig 4.1 The visualization of different prompting techniques by Yao et al [18] Each rectangle box represents a thought, which is a coherent language sequence that serves as an intermediate step toward problem-solving.

Chain-of-Thought (CoT) prompting, and Tree-of-Thought (ToT) prompting are both techniques used to enhance the problem-solving capabilities of large language models. Here, we discuss their key similarities and differences:

Similarities:

1. Enhanced reasoning: Both methods aim to improve the model's reasoning process by breaking down complex problems into smaller steps.

2. Intermediate steps: Both techniques encourage the model to show its work and produce intermediate reasoning steps. They both work to foster thoughts on step-by-step-based logic.

3. Improved performance: Both methods generally lead to better results on complex tasks compared to standard prompting.

Differences:

1. Structure:

CoT is linear, following a single path of reasoning from start to finish.

ToT is branching, exploring multiple possible reasoning paths simultaneously.

2. Exploration:

CoT typically produces one solution path.

ToT generates and evaluates multiple solution candidates, potentially finding better solutions.

3. Complexity:

CoT is simpler to implement and interpret.

ToT is more complex, requiring mechanisms to generate, evaluate, and prune different branches.

4. Adaptability:

CoT is less flexible once a reasoning path is chosen.

ToT can backtrack and explore alternative paths if a particular branch proves unfruitful.

5. Resource usage:

CoT is generally less computationally intensive.

ToT requires more computational resources to manage multiple branches.

6. Applicability:

CoT works well for problems with a clear, linear solution path.

ToT is particularly useful for problems with multiple possible approaches or solutions.

4.4 Tree-of-Thought on LLMs with Smaller Parameters

4.4.1 Address Current Issues in ToT

Despite its promising advancements, the Tree-of-Thought (ToT) framework has several limitations. One of the most significant challenges is its computational complexity. The primary source of computational complexity in ToT is the exponential growth of the search space as the problem complexity increases. Each node in the thought tree represents a potential intermediate state or partial solution, and each branch represents a different path that can be taken from that node. As the number of nodes and branches increases, the total number of potential paths that the system must consider can grow exponentially. This leads to a combinatorial explosion, especially in problems that require many steps or have multiple viable paths. The result is that the method requires substantial amount of time to execute, and a massive amount of token consumption and token generation. For example, the prompt token and generation token consumption were around 30,000 and 70,000 respectively solving a game of 24 in Yao's experiment. Using GPT-4, the cost of solving each game is around 1 dollar based on the pricing of OpenAI API. On the other hand, standard prompting costs less than 100 tokens to generate an answer for the game of 24. The immense difference between the two methods is shocking and presents a huge concern about the cost-effectiveness of ToT prompting method. Even Yao himself mentioned that deploying the solver on a free, open-sourced model would significantly reduce the cost of OpenAI API. [18]

Additionally, the effectiveness of ToT is highly dependent on the quality and accuracy of the intermediate steps generated by the language model. If the model frequently produces erroneous or irrelevant intermediate steps, the system might waste resources exploring unproductive paths. This is intriguing as scaling effect on model parameters can bring significant performance down.

These limitations suggest that while ToT is a powerful tool for enhancing reasoning in LLMs, further research is needed to fully understand its computational demands and dependency on LLMs. Therefore, in my research, I aim to address these problems by introducing the free, open-sourced, small-parameter Llama language model to the ToT prompting method.

For the experiment, we replicated three previous ToT experiments using open-sourced, smaller parameter models: Llama3-8b, which has 8 billion parameters, and Llama2-7b, which has 7 billion parameters. Our experiment focuses on three tasks:

1. Game of 24:

This task involves 100 games of 24 problem sets, which is a mathematical puzzle where the objective is to manipulate four integers with basic arithmetic operations to achieve a result of 24. We employ three different prompting strategies. (IO, CoT and ToT)

2. Sudoku Tasks:

This task involves solving Sudoku puzzles of varying sizes (3x3, 4x4, and 5x5 grids). We use four different prompting strategies (zero-shot, one-shot-CoT, few-shot-CoT, and ToT) to test the models

3. Creative Writing Task:

This task examines the natural language generation capability by introducing the creative writing task originally conducted in Yao’s research. The goal is to assess the model's ability to generate coherent and creative narratives based on given prompts, comparing the quality and creativity of the output using different prompting strategies.

To ensure a robust and controlled evaluation, we use the open-source framework Ollama to execute the ToT code and run Llama2-7b and Llama3-8b respectively, locally on our machine. This setup allows us to directly compare the performance of smaller parameter models under the same experimental conditions as larger models used in previous research.

4.4.2 Experimental Design and Methodology

Our experimental design is structured to meticulously analyze how smaller parameter models perform across different tasks and prompting strategies. The chosen tasks represent a mix of mathematical reasoning, logical deduction, and creative language generation, providing a comprehensive evaluation of the models' capabilities. For comparison between different language models, we'll employ the experiment data from original experiments conducted by Yao et al and Long. [18][19]

For the Game of 24 task, we evaluate each model's ability to solve 100 different instances of the game using the three prompting strategies mentioned. The performance metrics will be the success rate, which is the percentage of correctly solved problems. We'll compare our success rate on Llama3-8b with the success rate in Yao's experiment on GPT-4.

For the sudoku task, we implement the same Tree-of-Thought (ToT) based Sudoku solver experimented in Long's experiment. [19] The Tree-of-Thought (ToT) based Sudoku solver is a sophisticated tool designed to tackle Sudoku puzzles by utilizing a generic AI framework with specific enhancements for Sudoku. Users can input Sudoku puzzles in a natural language format, such as, "Please solve this 4x4 Sudoku puzzle [[3,*,*,2],[1,*,3,*],[*,1,*,3],[4,*,*,1]] where * represents a cell to be filled". This makes the system user-friendly and accessible to those who may not be familiar with traditional Sudoku notation. The solver consists of three main modules: the Checker Module, which verifies the correctness of each move based on Sudoku rules (each row, column, and subgrid must contain unique numbers) using a rule-based implementation; the Memory Module, which stores conversation history and maintains a search tree of all partially filled Sudoku boards using a data structure to keep track of different board states generated during the problem-solving process, enabling the solver to backtrack and explore alternative solutions; and the ToT Controller, which manages the overall search process, including decision-making and backtracking, using a rule-based implementation to direct the flow of solving strategies, deciding when to pursue a path further or when to revert to a previous state in the search tree to try a different approach. For the Sudoku tasks, we test the models on three different grid sizes, reflecting varying levels of complexity. The task was conducted on the solver, with each model tasked with solving multiple puzzles for each grid size using the four prompting strategies (zero-shot, one-shot-CoT, few-shot-CoT, and ToT). We produce a success rate for tasks on each prompt-grid size pair for analysis. Similarly, we'll compare our success rate on Llama3-8b with the success rate in Long's experiment on GPT-3.5-turbo.

For the Creative Writing task, we use a set of standardized prompts to generate narratives. The outputs are then evaluated based on coherence by producing a coherency score by the LLM. We compare the outputs generated using different prompting strategies to assess the impact on narrative quality. This is measured by introducing an autonomous comparison prompt to identify the best prompting strategies in each creative writing game and we keep a record of how many times each prompting methods generates the best response. We'll compare our data with existing data from

Yao’s experiment for analysis. [18]

4.4.3 Analysis and Expected Outcomes

The analysis focuses on identifying the strengths and limitations of smaller parameter models when using the ToT framework. We expect to see variations in performance across different tasks and prompting strategies. Specifically, we aim to determine whether the tree-like exploration and backtracking inherent in ToT can compensate for the reduced parameter size and how effectively smaller models can utilize this framework.

In this experiment, we hypothesize that:

Game of 24 and Sudoku tasks: ToT prompting will outperform traditional few-shot and CoT prompting, even with smaller models, due to its ability to explore multiple reasoning paths and correct errors through backtracking.

Creative Writing task: While ToT may enhance logical tasks, its impact on creative writing might be less pronounced, as narrative generation relies heavily on the richness of the language model, which may be constrained by smaller parameter sizes.

5. Experiment

5.1 Game of 24

The game of 24 was originally conducted by Yao et al, in their experiment to test the arithmetic and logical capability of the ToT prompting method. [18]

5.1.1 Game Overview

The Game of 24 is a mental arithmetic challenge where the objective is to use four given numbers and the basic arithmetic operations (addition, subtraction, multiplication, and division) to produce the number 24. Each number must be used exactly once, and operations can be performed in any order. The task's complexity lies in finding the correct combination of operations and the sequence in which they are applied.

5.1.2 Experiment Setup

Data Collection:

Data Source: The data for the experiment was sourced from 4nums.com, a website that offers various Game of 24 challenges.

Difficulty Levels: The games are ranked by difficulty based on how long it takes humans to solve them.

Subset Selection: For the experiment, a subset of relatively hard games was chosen, specifically those indexed from 901 to 1,000. This selection was based on the assumption that these would be more challenging for the language model and thus provide a better test of its problem-solving capabilities.

Task Definition:

Objective: Each task involves generating a valid equation that equals 24 using the four provided numbers, each exactly once.

Evaluation: A task is considered successful if the equation produced correctly equals 24.

5.1.3 Methodology

In this experiment, we are going to test various prompting and problem-solving techniques using the Llama2-7b running locally on the computer. Here are the prompting methods listed:

1. Input-Output (IO) Prompting:

Description: This is the simplest form of prompting, where the model is given a direct input and expected to generate the correct output.

Prompt Structure: The prompt included five in-context examples of the Game of 24, each showing the input numbers and the resulting equation.

Prompt Sample:

Use numbers and basic arithmetic operations (+ - * /) to obtain 24. The input and examples are provided. Generate only the answer to the input and nothing else.

Input: 4 4 6 8

Answer: (4 + 8) * (6 - 4) = 24

Input: 2 9 10 12

Answer: 2 * 12 * (10 - 9) = 24

Input: 4 9 10 13

Answer: (13 - 9) * (10 - 4) = 24

Input: 1 4 8 8

Answer: (8 / 4 + 1) * 8 = 24

Input: 5 5 5 9

Answer: 5 + 5 + 5 + 9 = 24

Input: 4 9 10 13

2. Chain-of-Thought (CoT) Prompting:

Description: This method involves breaking down the problem into a series of intermediate steps, or "thoughts," that lead to the final solution.

Prompt Structure: The input-output pairs are supplemented with three intermediate equations, each using two of the remaining numbers.

Prompt Sample:

Use numbers and basic arithmetic operations (+ - * /) to obtain 24. Each step, you are only allowed to choose two of the remaining numbers to obtain a new number. The input and examples are provided. Generate only the answer to the input and nothing else.

Input: 4 4 6 8

Steps:

4 + 8 = 12 (left: 4 6 12)

6 - 4 = 2 (left: 2 12)

2 * 12 = 24 (left: 24)

Answer: (6 - 4) * (4 + 8) = 24

Input: 2 9 10 12

Steps:

12 * 2 = 24 (left: 9 10 24)

10 - 9 = 1 (left: 1 24)

24 * 1 = 24 (left: 24)

Answer: (12 * 2) * (10 - 9) = 24

Input: 4 9 10 13

Steps:

13 - 10 = 3 (left: 3 4 9)

9 - 3 = 6 (left: 4 6)

4 * 6 = 24 (left: 24)

Answer: 4 * (9 - (13 - 10)) = 24

Input: 1 4 8 8

Steps:

8 / 4 = 2 (left: 1 2 8)

1 + 2 = 3 (left: 3 8)

3 * 8 = 24 (left: 24)

Answer: (1 + 8 / 4) * 8 = 24

Input: 5 5 5 9

Steps:

5 + 5 = 10 (left: 5 9 10)

10 + 5 = 15 (left: 9 15)

15 + 9 = 24 (left: 24)

Answer: ((5 + 5) + 5) + 9 = 24

Input: 4 9 10 13

3. Tree of Thoughts (ToT):

Description: This method involves deliberate problem-solving where the process is broken into multiple intermediate steps, allowing for backtracking and exploration of different reasoning paths.

Steps:

1. Thought Decomposition: The problem is decomposed into three intermediate steps.

2. Thought Generation: Possible next steps are proposed for each intermediate state.

3. State Evaluation: States are evaluated based on their potential to lead to a solution.

4. Search Algorithm: A breadth-first search (BFS) with a beam width of 5 was used to systematically explore the tree of thoughts.

Breadth Limit: The beam width (b) was set to 5, meaning that the top 5 candidate states were kept for further evaluation at each step.

In step 2, a prompt for the generation of ideas is formatted like this:

Input: 2 8 8 14

Possible next steps:

2 + 8 = 10 (left: 8 10 14)

8 / 2 = 4 (left: 4 8 14)

14 + 2 = 16 (left: 8 8 16)

2 * 8 = 16 (left: 8 14 16)

8 - 2 = 6 (left: 6 8 14)

14 - 8 = 6 (left: 2 6 8)

14 /   2 = 7 (left: 7 8 8)

14 - 2 = 12 (left: 8 8 12)

Input: 4 9 10 13

Possible next steps:

In step 3, an example prompt for evaluation is formatted like this:

Evaluate if given numbers can reach 24 (sure/likely/impossible)

!!!!!!!Generate only in the designated format as listed below, You DO NOT need to generate anything else!!!!!!!!!!!

10 14

10 + 14 = 24

sure

11 12

11 + 12 = 23

12 - 11 = 1

11 * 12 = 132

11 / 12 = 0.91

impossible

4 4 10

4 + 4 + 10 = 8 + 10 = 18

4 * 10 - 4 = 40 - 4 = 36

(10 - 4) * 4 = 6 * 4 = 24

sure

4 9 11

9 + 11 + 4 = 20 + 4 = 24

sure

5 7 8

5 + 7 + 8 = 12 + 8 = 20

(8 - 5) * 7 = 3 * 7 = 21

I cannot obtain 24 now, but numbers are within a reasonable range

likely

5 6 6

5 + 6 + 6 = 17

(6 - 5) * 6 = 1 * 6 = 6

I cannot obtain 24 now, but numbers are within a reasonable range

likely

10 10 11

10 + 10 + 11 = 31

(11 - 10) * 10 = 10

10 10 10 are all too big

impossible

1 3 3

1 * 3 * 3 = 9

(1 + 3) * 3 = 12

1 3 3 are all too small

impossible

{input (possible next step)}

For the IO and the CoT prompting, we used Llama2-7b to test each game once for all 100 games. And due to the limitations of our GPU and the fact that ToT prompting here demands huge amounts of token generation, we used the existing trial of Llama2-7b on ToT by csjiet on the GitHub repository tree-of-thoughts-with-llama (https://github.com/csjiet/tree-of-thoughts-with-llama).

Performance Metrics:

Primary Metric: Success rate across 100 games.

Calculation: The success rate was determined by the proportion of games where a valid solution was found.

Yao et al [18] also tested those same benchmarks using GPT-4, which is one of OpenAI's latest large language model series. GPT-4 has a parameter of 1.76 trillion parameters or 1760 billion parameters according to some resources. We compare Yao et al's GPT-4 results with my Llama2-7b results.

5.1.4 Experimental Results:

Success rate (Percent)		
Prompt Method/LLMs	GPT4-1760b (Yao et al [18])	Llama2-7b
Standard, Input-Output	7.3	4
Chain-of-Thought	4	0
Tree-of-Thought	74	0

Table 5.1 The result for the game of 24 task.

The experiment shows a huge decay in the success rate of the Tree-of-Thought from 74 percent to 0. While standard, IO prompting showed a decay of 3.3 percent from 7.3 percent to 4 percent. The CoT was also dropped to 0. In Llama2-7b, the standard, IO prompting performed the best while the CoT and the ToT prompting completely failed the task with a success rate of zero. This result is the opposite of the implementation on GPT4 where ToT prompting recorded a success rate dramatically greater than the other two prompting methods.

5.1.5 Discussion:

The experimental results reveal significant drop in performance between from GPT-4 to Llama2-7b across different prompting methods for the Game of 24 task. These findings warrant a deeper analysis of the factors contributing to these differences and their implications for language model capabilities and prompting techniques. First, the stark contrast in performance, especially for the IO and CoT method, can be largely attributed to the vast difference in model size. GPT-4, with its reported 1.76 trillion parameters, significantly outperforms Llama2-7b, which has only 7 billion parameters. This substantial gap in parameter count likely translates to GPT-4's superior ability to handle complex reasoning tasks and utilize advanced prompting techniques effectively. From GPT4 to Llama2-7b, Interestingly we observed Input-Output (IO) Prompting showed the smallest performance gap between the two models. This suggests that for straightforward prompts, even smaller models can achieve some success, albeit limited. The complete failure of Llama2-7b with CoT prompting, compared to GPT-4's modest success, indicates that smaller models may struggle to effectively utilize intermediate reasoning steps. This could be due to limitations in working memory or the ability to maintain coherent thought chains, which is a result consistent with the previous findings. The dramatic success of ToT with GPT-4 (74% success rate) contrasted with Llama2-7b's failure highlights the method's potential, but also its dependency on model capability. ToT's complexity, involving thought decomposition, generation, evaluation, and search, appears to be beyond the capabilities of smaller models like Llama2-7b. This was proven as Llama3-8b fails to produce executable next step and provide logical evaluation for each step. The results suggest that advanced prompting techniques like ToT may not scale down effectively to smaller models. This raises questions about the minimum model size or capability required to benefit from such sophisticated methods, similar to the limitations of CoT. Overall, the experiment results underscores the importance of continual research in both model scaling and the development of more universally applicable prompting and reasoning strategies in AI.

5.2 Sudoku

The sudoku experiment was originally conducted by Long [19] in this paper on the findings of Tree-of-Thought.

5.2.1 Overview

The paper by Long [19] on the Tree-of-Thought, on the other hand, presents a Sudoku puzzle solver based on the Tree-of-Thought (ToT) framework. This framework enhances large language models (LLMs) with additional agents and memory modules to improve their performance on mathematical problem-solving tasks. The primary goal is to explore the potential of ToT in solving generalized Sudoku problems, which are known to be NP-complete. The success of this approach indicates its capability to handle a wide range of mathematical and logical reasoning tasks.

5.2.2 Implementation Details

Prompter Agent:

Function: Guides the LLM by providing structured prompts.

Implementation: Uses a template where the problem description is the initial Sudoku board configuration, and the partial solution summary is the current node in the search tree. This helps in systematically exploring the solution space.

LLM Utilized:

In the original experiment by Long [19], the "OpenAI GPT-3.5-turbo" model, which is the model used in ChatGPT, with a parameter number of 20b, was used with a temperature parameter set to 1. We compare the data generated from Long [19] to perform our analysis

In our experiment, We used the MetaAI 'Llama3-8b' with a parameter number of 8b, the temperature, as well as other conditions, are set to the same.

5.2.3 Experimental Setup

1. Solver Variants:

Zero-Shot Solver (zs):

Approach: Directly posts the puzzle description to the LLM without additional context or examples.

Use Case: Suitable for evaluating the raw problem-solving capability of the LLM.

Example: Please solve this Sudoku puzzle [["1", "*", "*"], ["*", "1", "*"], ["*", "2", "*"]] where * represents a cell to be filled in. Please return your solution in the following JSON format: { "rows": [] }.

One-Shot Solver (os):

Approach: Provides a step-by-step solution to a 3x3 Sudoku puzzle as a single example before presenting the target puzzle.

Use Case: Helps the LLM understand the problem structure through a simple example.

Example: Here is an example showing how to solve a 3x3 Sudoku puzzle [[*, 3, 1], [*, 2, 3], [3, *, 2]]. First, notice that the only missing number in the first row is 2, so we can fill in the first cell in the first row with 2. Similarly, the first cell in the second row should be 2. Finally, the only missing number in the second column is 1. Hence, we can fill that cell with 1.In conclusion, the puzzle solution is [[2, 3, 1], [1, 2, 3], [3, 1, 2]]. Now, please solve this Sudoku puzzle [["1", "*", "*"], ["*", "1", "*"], ["*", "2", "*"]] where * represents a cell to be filled in. Please return your solution in the following JSON format: { "rows": [] }.

Few-Shot Solver (fs):

Approach: Provides multiple solved Sudoku examples in a Chain-of-Thought (CoT) format, showcasing step-by-step reasoning.

Use Case: Enhances the LLM's problem-solving capability by demonstrating the process of logical reasoning over several examples.

Example: Here is an example showing how to solve a 3x3 Sudoku puzzle [[*, 3, 1], [*, 2, 3], [3, *, 2]]. First, notice that the only missing number in the first row is 2, so we can fill in the first cell in the first row with 2. Similarly, the first cell in the second row should be 2. Finally, the only missing number in the second column is 1. Hence, we can fill that cell with 1. In conclusion, the puzzle solution is [[2, 3, 1], [1, 2, 3], [3, 1, 2]]. Here is another example showing how to solve Sudoku puzzle [[1, *, *], [*, 1, *], [*, 2, *]]. First, notice that the second column already has 1 and 2, so the first cell in the second row needs to be 3. After this step, the first row has 1 and 3. Hence the last cell in the first row must be 2. Now, look at the cell at the intersection of the second row and the third column. It must be 3. As a result, the cell at the intersection of the third row and the third column must be 1. The remaining cells are now easy to fill in. In conclusion, the puzzle solution is [[1, 3, 2], [2, 1, 3], [3, 2, 1]]. Now, please solve this Sudoku puzzle [["1", "*", "*"], ["*", "1", "*"], ["*", "2", "*"]] where * represents a cell to be filled in. Please return your solution in the following JSON format: { "rows": [] }.

Tree-of-Thought Solver (tot):

Approach: Utilizes the ToT framework, which incorporates the following key components:

Search Tree: Builds a tree where each node represents a partially filled Sudoku board.

Backtracking: Systematically explores different branches of the search tree, backtracking when a branch does not lead to a solution.

Heuristic Guidance: Uses heuristics to prioritize promising branches and prune unproductive paths.

LLM Interaction: The LLM generates potential moves, which are evaluated by the ToT controller, and the search tree is updated accordingly. An example prompt to interact with LLM would be: ‘Here is an example showing how to solve a 3x3 Sudoku puzzle [[*, 3, 1], [*, 2, 3], [3, *, 2]]. First, notice that the only missing number in the first row is 2, so we can fill in the first cell in the first row with 2. Similarly, the first cell in the second row should be 2. Finally, the only missing number in the second column is 1. Hence, we can fill that cell with 1.In conclusion, the puzzle solution is [[2, 3, 1], [1, 2, 3], [3, 1, 2]]. " Please try to solve this Sudoku puzzle [[\'1\', \'*\', \'*\'], [\'*\', \'1\', \'*\'], [\'*\', \'2\', \'*\']]. In the next solution you return, please just fill in a few cells since we will work together to solve the puzzle in multiple rounds of conversation. Please return your solution in the following JSON format: { "rows": [] }’

Exit condition: ToT exit the task and determine it fail when the solver was unable to solve in 100 rounds of conversations, or API calls.

2. Benchmarks:

The constructed benchmarks include ten puzzles each of sizes 3x3, 4x4, and 5x5.

Objective: To fill the n×n Sudoku grid with digits such that each row, column, and subgrid (if applicable) contains all digits from 1 to n.

3. Evaluation Metrics:

The success rate measures the fraction of puzzles in a benchmark set that are successfully solved by a solver. The success rates for each LLM-prompt method pair were recorded, with a maximum possible success rate of 1.0. For example, if a solver solves 4 out of 10 puzzles, the success rate is 0.4.

5.2.4 Experimental Results

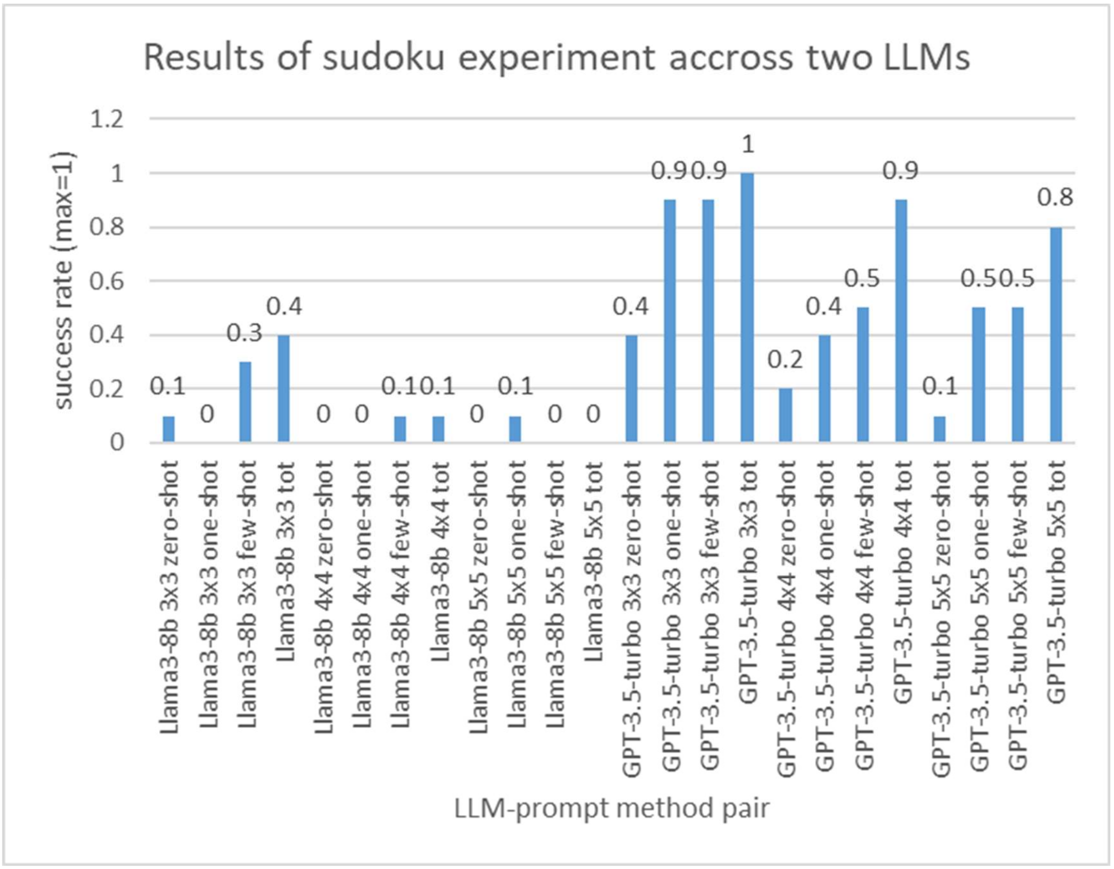

Figure 5.1 The result for the sudoku task on Llama3-8b. The experiment result from Long [19] on GPT-3.5-turbo was also included as reference.

Our experiment implemented the sudoku puzzle challenge using the Llama3-8b large language model. While the GPT-3.5-turbo model consistently outperformed the Llama3-8b model across all puzzle sizes and solver variants, the Llama3-8b model showed its best performance with the tot-shot solver on 3x3 puzzles, achieving a success rate of 0.4. In llama3-8b, the zero-shot approach yields very low success rates across all puzzle sizes, with the highest being 0.1 for 3x3 puzzles. And one-shot prompts show some improvement over zero-shot, especially for 5x5 puzzles, with a success rate of 0.1, the highest in 5x5 puzzle games. The few-shot approach yields the best results for Llama3-8b on 3x3 puzzles, with a success rate of 0.3. However, it still fails to solve 4x4 and 5x5 puzzles effectively, with success rates of 0.1 and 0.0, respectively. The ToT solver shows limited effectiveness for Llama3-8b; after recording the highest performance in 3x3 puzzles, the solver was not able to make another new best record with a success rate of 0.1 for 4x4 puzzles and 0 for 5x5 puzzles. Overall, we did not observe the performance boom in ToT solver across all sudoku games in the Llama3-8b model.

5.2.5 Discussion

The decay in standard prompt in all sudoku games from GPT-3.5-turbo to Llama3-8b is expected because we are using a language model with less than half of the parameters of GPT-3.5-turbo, consistent to the scaling law of learned models. The performance degradation is more pronounced as puzzle size increases, indicating that smaller models struggle more with scaling to larger, more complex problems. This aligns with the general understanding that larger models are better equipped to handle tasks of increasing complexity. While ToT showed remarkable improvements for GPT-

3.5-turbo, its benefits are much less pronounced for Llama3-8b. This casts doubt on the effectiveness of advanced reasoning frameworks like ToT may be dependent on the base model's capabilities. Smaller models might lack the capacity to fully leverage the potential of such frameworks. Both ToT and few-shot CoT performed better than zero-shot in 3x3 sudoku game suggests there might be a thought threshold for Llama3 to be able to handle those complicated tasks, and such threshold is reflected in the size of the sudoku grid. In simpler words, the 'thoughts' in ToT and CoT works in balancing three numbers in sudoku but not 4 or 5 or even more numbers in Llama3-8b. The possibility of such 'threshold' might be a future direction of research to delve into the potential of prompts and prompt engineering. On the other hand, there might be a simpler explanation for the slightly better performance in 3x3 sudoku, which is the fact that smaller grid size greatly increases the chance for the model to generate the correct answer. The language model stands a greater chance to guess the correct answer. For example, in 3x3 sudoku, if you have 3 grids of known numbers, and 6 empty grids to fill, there are overall 3 to power of 6 combinations or 729 combinations. Meanwhile, the 4x4 sudoku with 6 empty grids have 4096 possible combinations. Probability, in this case, can greatly boost the success rate in 3x3 sudoku games.

5.3 Creative Writing

5.3.1 overview

While previous tasks focused on games, which are good measures of logical reasoning, we haven't fully explored how well ToT prompting can handle the complexities of natural language. The creative writing task is the second experiment conducted by Yao et al, and it is an interesting benchmark for testing natural language generation. [18] We'll start with four random sentences. The input is 4 random sentences and the output should be a coherent passage with 4 paragraphs that end in the 4 input sentences respectively. The task aimed to assess creativity through written expression, evaluating aspects like originality, coherence, and adherence to the prompt.

5.3.2 Experiment Setup

To do this experiment, Yao et al [18] created a set of 100 writing games by randomly picking sentences from a website (randomwordgenerator.com). There wasn't a pre-written "perfect" answer for each prompt. Due to the limitations of our GPU processing power,

In this experiment, We used Llama3-8b model, and in the original experiment was conducted with GPT-4 which has1760 billion parameters. I'll only conduct 30 of them in our experiment due to the limitations of our GPU. We will use the data from the original experiment as reference for our analysis.

The experiment focused on how well the generated stories flowed. We used several methods to evaluate this:

1. Automated Scoring: We used prompt (zero-shot) to give each story a score from 1 to 10 for coherency determined by the LLMs. We generate one coherency score for each text generation, that is every game and prompting methods gets a coherency score. The scoring was conducted for the 30 different games that I've tried in our experiment. And in the

experiment of Yao et al [18], the scoring was tested for all 100 games.

2. Automated Comparing (Ranking): Besides scoring, a comparison between the outputs of different prompts is also an important benchmark. To address this, We wrote a prompt to let the language model compare the text generated from different prompts and request an output of ranking from the best to the worst. We then record which prompting method generates the best text in each game. We tried this comparing method on all 30 different creative writing game that We have experimented with. This comparing method was mainly served to replace the upcoming human evaluation method which We do not have resource to access this method.
3. Human evaluation: This was conducted by Yao et al. [18] Simply let humans decide which text generated is the best. Having human authors anonymously compare the flow of stories generated by different prompting methods. To avoid bias, the order of the stories was randomized for each comparison. This was done for all 100 games of creative writing.

Prompting Methods:

Since the task requires creativity, our methods do not involve pre-training the model (zero-shot learning).

Simple Instruction Prompt (IO): The IO method tells the model to directly create a coherent passage that follows the given prompt. An example of an IO prompt is: ‘Write a coherent passage of 4 short paragraphs. You should strictly follow the requirement. The requirement is that the end sentence of each paragraph must be each of the following four sentences: It isn't difficult to do a handstand if you just stand on your hands. It caught him off guard that space smelled of seared steak. When she didn’t like a guy who was trying to pick her up, she started using sign language. Each person who knows you has a different perception of who you are.’

Think-Then-Talk Prompt (CoT): The CoT method is more involved. It first asks the model to come up with a short plan, then use that plan to write the passage. It's like the model has a brainstorming session before writing. An example of a CoT prompt is: ‘Write a coherent passage of 4 short paragraphs. You should strictly follow the requirement. The requirement is that the end sentence of each paragraph must be each of the following four sentences: It isn't difficult to do a handstand if you just stand on your hands. It caught him off guard that space smelled of seared steak. When she didn’t like a guy who was trying to pick her up, she started using sign language. Each person who knows you has a different perception of who you are.

Make a plan then write. Your output should be of the following format:

Plan:

Your plan here.

Passage:

Your passage here.’

Think-Then-Talk Prompt (ToT): This is a solver with a depth of 2 with the following elements:

Plan Generation: The model comes up with 3 different plans for the story based on the prompt. It then uses a simple prompt (like a vote) to analyze these plans and choose the one that seems most promising. Three votes are taken, and the average was counted as the overall vote. Note that the original experiment was able to come up with 5 different plans and 5 votes taken average. We conduct our experiment with those parameters shrunk to 3 due to GPU limitations.

Passage Generation: Based on the chosen plan, the model generates 5 different story passages. The prompt is the same as IO prompt as this stage. Then, it uses a simple vote prompt to pick the best passage out of the 5 it created. Then prompt for picking the best passage designed as the following: ‘Given an instruction and several choices, decide which choice is most promising. Analyze each choice in detail, then conclude in the last line "The best choice is {best choice}’

Key Points:

The model only considers one option (plan or passage) at each step (breadth limit of 1).

A simple prompt asking the model to analyze choices and pick the best one is used for voting in both steps (plan and passage selection).

5.3.3 Experiment results

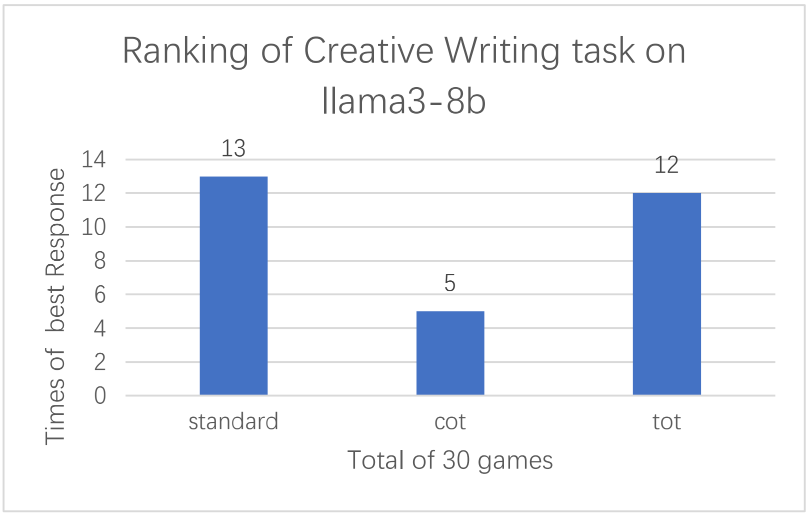

Figure 5.2 The results for the autonomus comparison task. Record numbers are the where each prompting method was the best out of all methods in a single creative writing game.

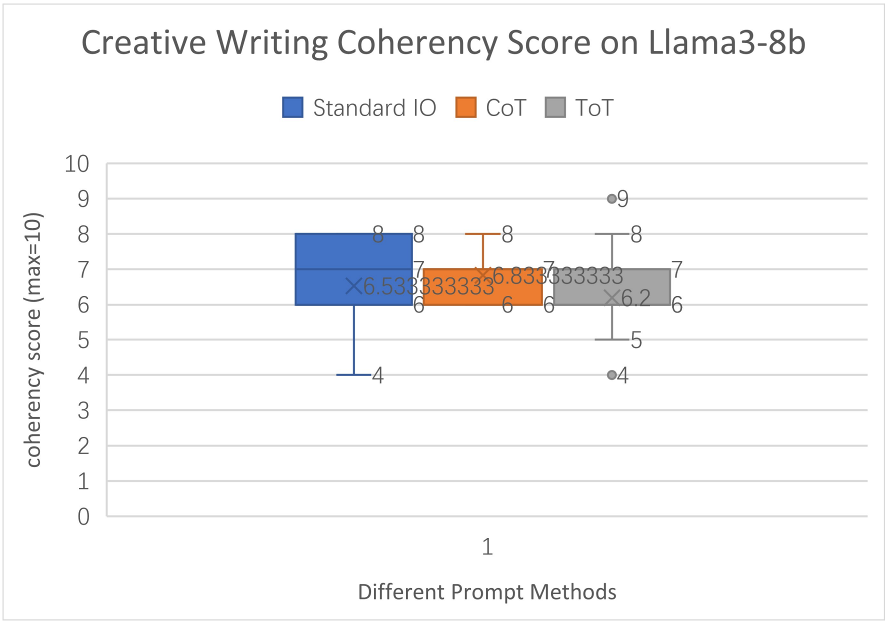

Figure 5.3 The coherency score of our creative writing experiment on Llama3-8b was presented using a box and whisker plot.

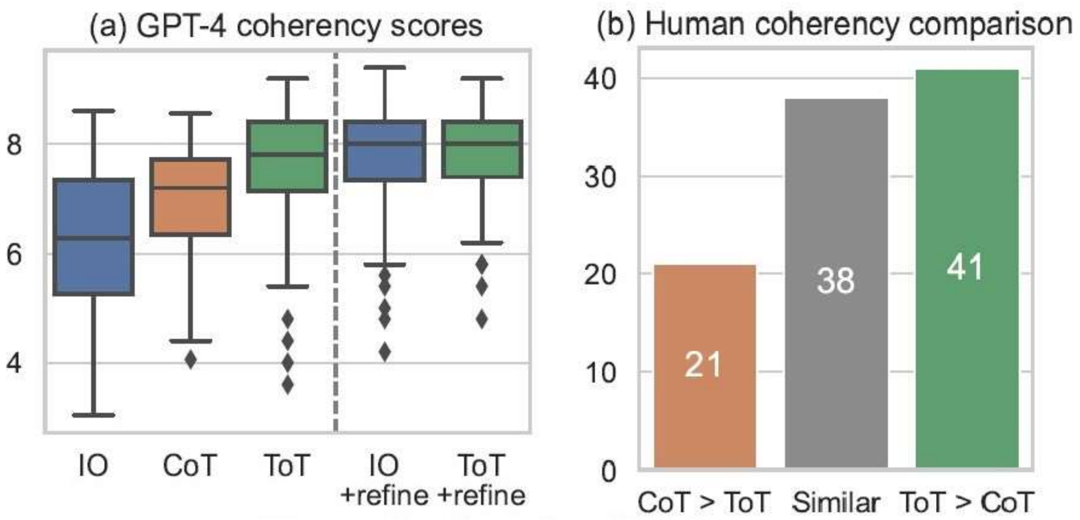

Figure 5.4 the original experiment by Yao et al on GPT-4. [18, Fig 5]

We produced the results for automated scoring and automated comparing in our creative writing task. We observe that all prompting methods are able to produce generally coherent result as most of the coherency scores are greater than 5 except rare occasions. CoT method was able to score the highest average coherency of 6.8 out of 10 while the other two as also similarly competent as the coherency score of IO and ToT prompting are not far from that number. Comparing to experiment conducted on GPT-4, where different prompting methods results in coherency score far from each other, we observe a closer average across different prompting methods in Llama3-8b. For the automated comparison, we observe that the numbers of best response were close for IO and ToT prompting, each receiving 13 and 12 best responses across the 30 games we experimented. On the other hand, CoT performed poorly in this benchmark with only 5 best responses recorded. Overall, we observe that performance of ToT in creative writing was not extraordinary compared to other prompting methods in the Llama3-8b model.

5.3.4 Discussion

The results of this creative writing experiment with the Llama3-8b model present an interesting contrast to the original study conducted with GPT-4. While the GPT-4 experiment showed significant differences between prompting methods, the Llama3-8b results indicate more similar performance across methods. The result could be a useful argument that the benefits of more complex prompting techniques like Chain of Thought (CoT) and Tree of Thoughts (ToT) may be more pronounced in larger, more advanced models. One possible interpretation is that smaller models may have a "ceiling effect" where their base capabilities limit the potential improvements from advanced prompting. This could explain why we see less differentiation between IO, CoT, and ToT methods in Llama3-8b compared to GPT-4. The discrepancy between coherency scores and automated comparisons for the CoT method is particularly intriguing. It suggests that different evaluation metrics may capture different aspects of text quality, and relying on a single metric could lead to incomplete conclusions. This highlights the importance of using multiple evaluation methods in natural language generation tasks. Another interesting aspect is the performance of the ToT method. While it didn't show extraordinary improvements over simpler methods in this case, it's worth considering whether the implementation of ToT could be optimized for smaller models. Perhaps adjusting the depth or breadth of the "tree" could yield better results in models with fewer parameters. We also need to consider the cost-effective aspects of ToT as it utilizes much more tokens than standard IO prompting. These findings also underscore the need for more research into how prompting techniques scale across different model sizes and architectures. It's possible that certain prompting methods may need to be adapted or reimagined for smaller models to achieve optimal performance. Future research in this area could benefit from exploring a wider range of model sizes, fine-tuning prompting techniques for different model scales, and incorporating human evaluation alongside automated metrics. This could provide a more comprehensive understanding of how to optimize language model performance across various tasks and model sizes.

Lastly, it's important to note the limitations of automated evaluation in creative tasks. While automated metrics conducted in our experiment provide valuable insights, they may not fully capture the nuances of creative writing. This experiment's lack of human evaluation (which was present in the original study) leaves open questions about how these generated texts might be perceived by human readers.

6. Conclusions and further research

Our experiments conducted on the Tree-of-Thought (ToT) method with smaller parameter models such as Llama2-7b and Llama3-8b reveal significant insights into the limitations and potential of current natural language processing (NLP) techniques for long-range reasoning tasks. The following discussion will delve into the critical analysis of these findings, the challenges encountered, and the implications for future research.

6.1 Performance Comparison

The experiments on the Game of 24 and Sudoku puzzles highlight a clear performance disparity between smaller parameter models and larger models like GPT-4. The success rate for the Game of 24 using the ToT method plummeted from 74% with GPT-4 to 0% with Llama2-7b. Similarly, the Chain-of-Thought (CoT) method showed a complete drop to 0% success with Llama2-7b, compared to a modest success rate with GPT-4. The Sudoku puzzle experiments also demonstrated that the Llama3-8b model struggled significantly across all puzzle sizes, showing its best performance (0.4 success rate) only with the ToT solver on 3x3 puzzles.

These findings underscore the importance of model size in leveraging advanced prompting techniques. The substantial parameter gap between GPT-4 (1.76 trillion parameters) and Llama2-7b (7 billion parameters) translates to a stark difference in handling complex reasoning tasks. This suggests that the effectiveness of sophisticated methods like ToT is closely tied to the underlying model's capacity, reinforcing the necessity for large models to achieve meaningful improvements in problem-solving tasks.

In short, adding our research into the known prompt engineering system, the underlying mechanisms that make combinations of prompting and model size (like ToT with larger LLMs in the case of our research) effective can be summarized as follows:

Structured Exploration: ToT's ability to explore multiple paths and backtrack enhances its problem-solving capabilities, particularly for complex tasks that benefit from considering various approaches.

Error Correction: The structured decision-making process in ToT allows for better error correction and refinement of solutions, leading to more reliable outcomes.

Scalability: Larger models can handle the increased computational demands of ToT prompting, leveraging their extensive parameters to manage multiple reasoning branches effectively.

6.2 Challenges and Limitations

The experiments on the Tree-of-Thought (ToT) method with smaller parameter models such as Llama2-7b and Llama3-8b highlighted several significant challenges and limitations. These are crucial to understand for future advancements in natural language processing (NLP) and artificial intelligence (AI) research.

Computational Demands

One of the primary challenges encountered during the experiments was the computational demand associated with the ToT method. The ToT framework, which involves thought decomposition, generation, evaluation, and search, requires extensive token generation and processing. This high token consumption leads to substantial computational costs, particularly when running on models with limited resources like Llama2-7b. The necessity for a beam width in the breadth-first search algorithm further exacerbates this issue, as it increases the number of candidate states that need to be evaluated at each step.

Due to these high computational costs, extensive testing on smaller models was constrained by hardware limitations. This made it difficult to fully explore the potential of these models using the ToT method and limited the ability to optimize and refine the approach. Future research must address these computational challenges, possibly by developing more efficient algorithms or by leveraging distributed computing resources.

Model Capacity and Coherent Thought Chains

Another critical limitation observed was the smaller models' capacity to maintain coherent chains of thought and manage complex contextual relationships. The CoT and ToT methods, which rely on breaking down problems into intermediate steps and backtracking when necessary, require models to hold and process substantial amounts of intermediate information. Smaller models, such as Llama2-7b with only 7 billion parameters, struggle with these demands.

This limitation was evident in the experiments, where Llama2-7b showed a complete failure in CoT and ToT tasks. The inability to maintain coherent chains of thought and logical consistency over extended reasoning paths suggests that smaller models lack the necessary working memory and processing power. Addressing this limitation might involve improving the memory mechanisms within these models or developing techniques to enhance their contextual understanding without significantly increasing their size.

Thought Threshold Hypothesis

The 'thought threshold' hypothesis, observed particularly in the Sudoku experiments, suggests that smaller models may only effectively handle simpler problems. For example, while Llama3-8b showed some success in solving 3x3 Sudoku puzzles, it failed with more complex 4x4 and 5x5 grids. This implies a threshold in the model's capacity to balance a larger number of variables or maintain logical consistency over more complex problem spaces.

Understanding and addressing this threshold is critical for improving the performance of smaller models on complex tasks. Research should focus on identifying the exact point at which these models can no longer manage increasing complexity and developing strategies to extend their capabilities within these bounds. This might involve simplifying tasks through more effective prompt engineering or augmenting models with external memory systems.

Evaluation Metrics and Benchmarking

Another limitation encountered was the reliance on specific evaluation metrics and benchmarks that may not fully capture the nuances of model performance. Success rates based on the fraction of solved puzzles provide a straightforward measure, but they might not reflect the models' capabilities in dealing with partially correct solutions or their progress toward solving complex problems.

Developing more nuanced evaluation metrics that consider partial success, the quality of intermediate steps, and the models' ability to learn and improve over time could provide a more comprehensive understanding of their performance. Additionally, expanding the range of benchmarks to include a wider variety of tasks and problem complexities can help identify strengths and weaknesses in different models and prompting methods.

6.3 Implications for Future Research

The results from these experiments suggest several avenues for future research. First, there is a need to explore alternative prompting strategies or modifications to existing methods that could enhance the performance of smaller models. For example, optimizing the way intermediate steps are generated and evaluated in ToT could potentially reduce token consumption and improve efficiency.

Another promising direction is the development of hybrid models that combine the strengths of large and small models. By leveraging the computational efficiency of smaller models for straightforward tasks and using larger models for more complex reasoning, it may be possible to achieve a balanced approach that mitigates the high costs associated with large-scale models.

Further research could also investigate the scalability of reasoning frameworks like ToT and CoT across different model sizes. Understanding the specific capabilities and limitations at various scales can help in designing more robust and adaptable NLP systems. This includes studying the minimum model size required to effectively utilize advanced prompting techniques and identifying potential enhancements that could extend the applicability of these methods to smaller models.

Additionally, the concept of 'thought threshold' warrants deeper investigation. Determining the exact point at which smaller models can no longer manage increasing complexity could inform the development of tailored strategies that maximize their efficiency within these bounds. This might involve innovative approaches to prompt engineering that simplify tasks without compromising the logical rigor required for accurate problem-solving.

6.4 Practical Applications and Long-Term Goals

The implications of this research extend beyond theoretical advancements, offering practical benefits for real-world applications. Enhancing the performance of smaller models in long-range reasoning tasks could make sophisticated NLP tools more accessible and cost-effective, particularly for organizations with limited computational resources. This democratization of advanced AI capabilities could drive innovation across various sectors, from education and healthcare to finance and customer service.

In the long term, the goal is to create NLP systems that can seamlessly integrate multiple reasoning strategies, adapting to the complexity of tasks dynamically. Achieving this would require ongoing collaboration between AI researchers, engineers, and domain experts to refine and validate new methods. By building on the findings of this research, the NLP community can move closer to developing versatile, scalable models that deliver consistent performance across a broad spectrum of applications.

6.5 Next Steps

Based on the insights gained from this study, several next steps are recommended to further explore and address the challenges identified:

1. Model Optimization: Investigate ways to optimize smaller models for specific tasks by refining prompting techniques and reducing token consumption in methods like ToT and CoT.

2. Hybrid Model Development: Explore the creation of hybrid models that can balance the computational demands and leverage the strengths of both large and small models.

3. Threshold Analysis: Conduct detailed studies to pinpoint the 'thought threshold' for different models and develop strategies to maximize efficiency within these limits.

4. Prompt Engineering: Innovate new approaches in prompt engineering that simplify task complexity without losing logical integrity, making advanced reasoning methods more accessible to smaller models.

5. Collaborative Research: Foster collaboration between AI researchers and industry practitioners to apply these findings in real-world contexts, ensuring practical relevance and impact.

By addressing these next steps, future research can build on the current study's findings to enhance the performance and applicability of NLP models in long-range reasoning tasks, ultimately contributing to the advancement of artificial intelligence as a whole.

7. References

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998-6008).

2. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.

3. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI blog, 1(8), 9.

4. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in neural information processing systems, 33, 1877-1901.

5. Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big?. In Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency (pp. 610-623).

6. Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and Policy Considerations for Deep Learning in NLP. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (pp. 3645-3650).

7. Floridi, L., & Cowls, J. (2019). A unified framework of five principles for AI in society. Harvard Data Science Review, 1(1).

8. Lu, J., Batra, D., Parikh, D., & Lee, S. (2019). ViLBERT: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. Advances in neural information processing systems, 32.

9. Ruder, S., Peters, M. E., Swayamdipta, S., & Wolf, T. (2019). Transfer learning in natural language processing. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials (pp. 15-18).

10. Holtzman, A., Buys, J., Du, L., Forbes, M., & Choi, Y. (2020). The Curious Case of Neural Text Degeneration.

11. Ziegler, D. M., Stiennon, N., Wu, J., Brown, T. B., Radford, A., Amodei, D., Christiano, P., & Irving, G. (2019). Fine-Tuning Language Models from Human Preferences.

12. OpenAI. (2023). "GPT-4 Technical Report." arXiv preprint arXiv:2303.08774v6.

13. Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., ... & Edunov, S. (2023). Llama 2: Open Foundation and Fine-Tuned Chat Models. Retrieved from [arXiv:2307.09288v2](https://arxiv.org/abs/2307.09288v2).

14. Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., & Iwasawa, Y. (2022). Large Language Models are Zero-Shot Reasoners. In Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS 2022).

15. Wei, J., Bosma, M., Zhao, V., Guu, K., Yu, A. W., Lester, B., ... & Chi, E. H. (2022). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv preprint arXiv:2201.11903v6.

16. Wu, T., Jiang, E., Donsbach, A., Gray, J., Molina, A., Terry, M., & Cai, C. J. (2022). PromptChainer: Chaining large language model prompts through visual programming [ArXiv preprint ArXiv:2203.06566].

17. Meta AI, Introducing Meta Llama 3: The most capable openly available LLM to date. [online] https://ai.meta.com/blog/meta-llama-3/ [Accessed 2024-06-20]

18. Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., & Narasimhan, K. (2023). Tree of Thoughts: Deliberate Problem Solving with Large Language Models. NeurIPS 2023, [ArXiv preprint ArXiv:2305.10601].

19. Long, J. (2023). Large language model guided tree-of-thought [ArXiv preprint

ArXiv:2305.08291].

Acknowledgments

I want to thank Professor Alex Fukunaga for this expert guidance, I met him in one of his classes on the introductory of machine learning and the class was very interesting and explanative, it was a huge honor to work with him and I look forward to having more opportunities to work with him in my future research career.

I also want to thank my peers in PEAK, whom many of them become my friends, although they are from different backgrounds, I couldn’t wish to have a better team than this to study with. We sometimes compete with each other but mostly we work together in groups for the exam or group assignments, my college life wouldn’t be that much fun without all of you.

Finally, I want to thank everyone I’ve been inspired by, my high school mates, my girlfriend, my parents, and friends that I’ve met both inside and outside UTokyo. It’s every interaction and every inspiration pieced together to make me who I am here and now. A big thank you to you all!