Graph of Thoughts: Solving Elaborate Problems with Large Language Models

Maciej Besta[*1], **Nils Blach**[*1], **Ales Kubicek**[1], **Robert Gerstenberger**[1], **Michał Podstawski**[2], **Lukas Gianinazzi**[1], **Joanna Gajda**[3], **Tomasz Lehmann**[3], **Hubert Niewiadomski**[3], **Piotr Nyczyk**[3], **Torsten Hoefler**[1]

[1]ETH Zurich
[2]Warsaw University of Technology
[3]Cledar
bestam@inf.ethz.ch, nils.blach@inf.ethz.ch, htor@inf.ethz.ch

Abstract

We introduce Graph of Thoughts (GoT): a framework that advances prompting capabilities in large language models (LLMs) beyond those offered by paradigms such as Chain-of-Thought or Tree of Thoughts (ToT). The key idea and primary advantage of GoT is the ability to model the information generated by an LLM as an *arbitrary graph*, where units of information ("LLM thoughts") are vertices, and edges correspond to dependencies between these vertices. This approach enables combining arbitrary LLM thoughts into synergistic outcomes, distilling the essence of whole networks of thoughts, or enhancing thoughts using feedback loops. We illustrate that GoT offers advantages over state of the art on different tasks, for example increasing the quality of sorting by 62% over ToT, while simultaneously reducing costs by >31%. We ensure that GoT is extensible with new thought transformations and thus can be used to spearhead new prompting schemes. This work brings the LLM reasoning closer to human thinking or brain mechanisms such as recurrence, both of which form complex networks.

1 Introduction

Large language models (LLMs) are taking over the world of AI. Recent years saw a rapid development of models primarily based on the decoder-only Transformer variant (Vaswani et al. 2017), such as GPT (Radford et al. 2018, 2019; Bubeck et al. 2023; Brown et al. 2020), PaLM (Chowdhery et al. 2022), or LLaMA (Touvron et al. 2023b).

Prompt engineering is a resource-efficient approach for solving different LLM tasks. In brief, one includes the task description within the input sent to an LLM. If this description is appropriately formulated, the LLM solves the task using its autoregressive token-based mechanism for generating text. Such prompts may contain example tasks with solutions (few-shot prompting, also referred to as in-context learning (ICL)), or even no example tasks at all (zero-shot prompting). In recent years it was shown that this mechanism can be used to solve a broad set of tasks that involve mathematical, commonsense, or symbolic reasoning.

Chain-of-Thought (CoT) (Wei et al. 2022) is an approach for prompting, in which one includes the intermediate steps of reasoning within the prompt (intermediate "thoughts"), besides the task input/output. CoT was shown to significantly improve the capability of LLMs to solve problems without resorting to any model updates. One major improvement over CoT, Self-Consistency with CoT (CoT-SC) (Wang et al. 2023b), is a scheme where multiple CoTs are generated, and then the best one is selected as the outcome. More recently, CoT and CoT-SC were extended with Tree of Thoughts (ToT) (Long 2023; Yao et al. 2023a; Xie et al. 2023), which models the LLM reasoning process with a tree. This facilitates using different paths of thoughts, and offers novel capabilities such as backtracking from non-promising outcomes. Unfortunately, the ToT approaches still fundamentally limit the reasoning abilities within a prompt by imposing the rigid tree structure on the thought process.

In this work, we argue that fundamentally more powerful prompting can be achieved by enabling LLM thoughts to form an arbitrary graph structure. This is motivated by numerous phenomena such as human reasoning, brain structure, or algorithmic execution. When working on a novel idea, a human would not only follow a chain of thoughts (as in CoT) or try different separate ones (as in ToT), but would actually form a more complex network of thoughts. For example, one could explore a certain chain of reasoning, backtrack and start a new one, then realize that a certain idea from the previous chain could be combined with the currently explored one, and merge them both into a new solution, taking advantage of their strengths and eliminating their weaknesses. Similarly, brains form complex networks, with graph-like patterns such as recurrence (Friston 2008). Executing algorithms also expose networked patterns, often represented by Directed Acyclic Graphs. The corresponding *graph-enabled transformations* bring a promise of more powerful prompting when applied to LLM thoughts, but they are not naturally expressible with CoT or ToT.

We observe that these (and many other) thought transformations can be naturally enabled when *modeling the reasoning process of an LLM as a graph*. For this, we propose Graph of Thoughts (GoT)[1], an approach that *enhances LLMs' capabilities through networked reasoning* (**contribution #1**). In GoT, an LLM thought is modeled as a vertex, while an edge is a dependency between such

[*]These authors contributed equally.

[1]Extended Technical Report: https://arxiv.org/abs/2308.09687

thoughts. Using GoT, one can aggregate arbitrary thoughts by constructing vertices that have more than one incoming edge. Overall, the graph abstraction harnessed by GoT seamlessly generalizes CoT and ToT to more complex thought patterns, *without resorting to any model updates*.

Yet, putting GoT to practice requires solving several design challenges. For example, what is the best graph structure for different tasks? How to best aggregate thoughts to maximize accuracy and minimize cost? To answer these and many other questions, we carefully design a modular architecture[2] for implementing GoT (**contribution #2**), coming with two design highlights. First, we enable a *fine-grained control over individual thoughts*. This enables us to fully control the ongoing conversation with the LLM, and apply advanced thought transformations, such as combining most promising thoughts from the ongoing reasoning into a new one. Second, we ensure that our architecture can be seamlessly extended with novel thought transformations, patterns of reasoning (i.e., graphs of thoughts), and LLM models. This enables rapid prototyping of novel prompting ideas using GoT, while experimenting with different models such as GPT-3.5, GPT-4, or Llama 2 (Touvron et al. 2023a).

We illustrate several use cases for GoT (sorting, keyword counting for summaries, set operations, document merging) and we detail how to implement them using the graph-based paradigm (**contribution #3**). We evaluate GoT and show its advantages over the state of the art (**contribution #4**). Overall, we observe that GoT is particularly well-suited for tasks that can be naturally decomposed into smaller subtasks that are solved individually and then merged for a final solution. Here, GoT outperforms other schemes, for example improving upon CoT and ToT by, respectively, $\approx$70% and $\approx$62%, in terms of the quality of sorting, while *simultaneously* reducing costs by $>$31% over ToT.

We qualitatively compare GoT to other prompting schemes[3] in Table 1. GoT is the only one to enable arbitrary graph-based thought transformations within a prompt, such as aggregation, embracing all previously proposed schemes.

Finally, we propose a new metric for evaluating a prompting strategy, the *volume of a thought* (**contribution #5**). With this metric, we aim to understand better the differences between prompting schemes. For a given thought v, the volume of v is *the number of LLM thoughts, from which one can reach v using directed edges*. Intuitively, these are all the LLM thoughts that have had the potential to contribute to v. We show that GoT, by incorporating thought transformations such as aggregation, enables thoughts to have fundamentally larger volumes than other schemes.

[2]Website & Code: https://github.com/spcl/graph-of-thoughts

[3]Note that we do not include a recent scheme called Graph-of-Thought (Yao, Li, and Zhao 2023) because it is not a prompting scheme. While its name suggests close connections to ToT and CoT, as a fine-tuning scheme, it resorts to model updates, and is thus outside the focus of this work. Similarly, the graph-of-thoughts repository (qrdlgit 2023) does not enable general graph-based reasoning and harnesses instead ToT with BFS.

Scheme	Sc?	Mc?	Tr?	Ag?
CoT	🔋	✖	✖	✖
CoT-SC	🔋	🔋	✖	✖
ToT (Xie et al. 2023)	🔋	🔋	🪫	✖
ToT (Long 2023)	🔋	🔋	🔋	✖
ToT (Yao et al. 2023a)	🔋	🔋	🔋	✖
GoT	🔋	🔋	🔋	🔋

Table 1: Comparison of prompting schemes, with respect to the supported transformations of thoughts. *"Sc?"*: single chain of thoughts? *"Mc?"*: multiple chains of thoughts? *"Tr?"*: tree of thoughts? *"Ag?"*: arbitrary graph of thoughts? "🔋": full support, "🪫": partial support, "✖": no support.

2 The GoT Framework

We now detail the GoT framework. We present it in Figure 1, and compare it to other prompting strategies.

The **conversation with the LLM** consists of user messages (*prompts*) and the LLM replies (*thoughts*). We follow the established notation (Yao et al. 2023a) and we denote a pre-trained language model (LM) with parameters θ as p_θ. Lowercase letters such as $x, y, z, ...$ indicate LLM thoughts.

Formally, GoT can be modeled as a tuple $(G, \mathcal{T}, \mathcal{E}, \mathcal{R})$, where G is the "LLM reasoning process" (i.e., all the LLM thoughts within the context, with their relationships), $\mathcal{T}$ are the potential thought transformations, $\mathcal{E}$ is an evaluator function used to obtain scores of thoughts, and $\mathcal{R}$ is a ranking function used to select most relevant thoughts.

2.1 Reasoning Process

We model the reasoning process as a directed **graph** $G = (V, E)$; V is a set of vertices and $E \subseteq V \times V$ is a set of edges. A vertex contains a *solution* to a problem at hand (be it an initial, intermediate, or a final one). The concrete form of such a thought depends on the use case; it could be a paragraph (in writing tasks) or a sequence of numbers (in sorting). A directed edge (t_1, t_2) indicates that thought t_2 has been constructed using t_1 as "direct input", i.e., by explicitly instructing the LLM to use t_1 for generating t_2.

We associate G with the LLM reasoning process. To advance this process, one applies **thought transformations** to G. An example of such a transformation is to merge best-scoring (so far) thoughts into a new one. Another example is to loop over a thought, in order to enhance it. Note that these transformations strictly extend the set of transformations available in the CoT, CoT-SC, or ToT.

2.2 Transformations of Thoughts

GoT enables novel transformations of thoughts thanks to the graph-based model for reasoning. We refer to them as **graph-enabled transformations**. For example, in writing, one could combine several input articles into one coherent summary. In sorting, one could merge several sorted subarrays of numbers into a final sorted array.

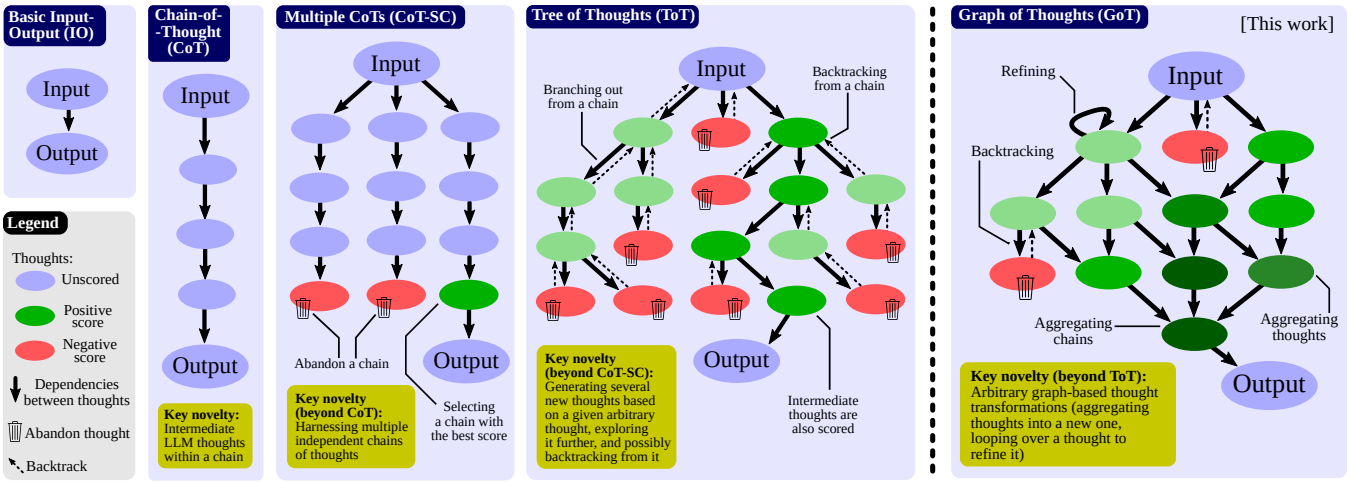

Figure 1: Comparison of Graph of Thoughts (GoT) to other prompting strategies.

Formally, each such transformation can be modeled as $\mathcal{T}(G, p_\theta)$ where $G = (V, E)$ is the graph reflecting the current state of the reasoning, and p_θ is the used LLM. $\mathcal{T}$ modifies G usually by adding new vertices and their incoming edges. We have $G' = \mathcal{T}(G, p_\theta) = (V', E')$, where $V' = (V \cup V^+) \setminus V^-$ and $E' = (E \cup E^+) \setminus E^-$. V^+ and E^+ are new vertices and edges inserted into G to model the new thoughts and their dependencies, respectively. To maximize the expressiveness of GoT – we also enable the user to explicitly *remove* thoughts, by specifying the corresponding vertices and edges to be removed (V^- and E^-, respectively). This enables seamless incorporation of schemes where, in order to save space within the context, one can remove parts of reasoning that do not promise improvements.

First, with GoT, one can **aggregate arbitrary thoughts** into new ones, to combine and reinforce the advantages of these thoughts, while eliminating their disadvantages. In the basic form, in which only one new vertex is created, $V^+ = \{v^+\}$ and $E^+ = \{(v_1, v^+), ..., (v_k, v^+)\}$, where $v_1, ..., v_k$ are the merged k thoughts. More generally, this enables **aggregating reasoning paths**, i.e., longer chains of thoughts, beyond just individual thoughts. With the graph model, it is simply achieved by adding outgoing edges from the vertices $v_1, ..., v_k$, modeling final thoughts in several chains, into a single thought v^+ combining these chains.

Another thought transformation is the **refining** of a current thought v by modifying its content: $V^+ = \{\}$ and $E^+ = \{(v, v)\}$. This loop in the graph indicates an iterated thought with the same connections as the original thought.

Finally, one can **generate one or more new thoughts based on an existing single thought** v. This class embraces analogous reasoning steps from earlier schemes, such as ToT or CoT-SC. Formally, we have $V^+ = \{v_1^+, ..., v_k^+\}$ and $E^+ = \{(v, v_1^+), ..., (v, v_k^+)\}$.

2.3 Scoring & Ranking Thoughts

Thoughts are scored to understand whether the current solution is good enough. A score is modeled as a general function $\mathcal{E}(v, G, p_\theta)$, where v is a thought to be evaluated. We use the state of the whole reasoning process (G) in $\mathcal{E}$ for maximum generality, because – for example – in some evaluation scenarios, scores may be relative to other thoughts.

GoT can also rank thoughts. We model this with a function $\mathcal{R}(G, p_\theta, h)$ where h specifies the number of highest-ranking thoughts in G to be returned by $\mathcal{R}$. While the specific form of $\mathcal{R}$ depends on the use case, we most often use a simple yet effective strategy where h thoughts with the highest scores are returned, i.e., $v_1, ..., v_h = \mathcal{R}(G, p_\theta, h)$.

Specific forms of $\mathcal{E}$ and $\mathcal{R}$ depend on the use case. We discuss the details in Section 4. For example, the score (or rank) for sorting corresponds to the count of elements correctly sorted (or incorrectly, when using the error as a score).

3 System Architecture & Extensibility

The GoT architecture consists of a set of interacting modules, see Figure 2 (the blue part). These modules are the Prompter (prepares the messages for the LLM), the Parser (extracts information from LLM thoughts), the Scoring module (verifies and scores the LLM thoughts), and the Controller (coordinates the entire reasoning process, and decides on how to progress it). The Controller contains two further important elements: the Graph of Operations (GoO) and the Graph Reasoning State (GRS). GoO is a static structure that specifies the *graph decomposition of a given task*, i.e., it prescribes transformations to be applied to LLM thoughts, together with their order & dependencies. GRS is a dynamic structure that maintains the state of the ongoing LLM reasoning process (the history of its thoughts and their states).

Prompter The Prompter prepares the prompts to be sent to the LLM. This module is responsible for the specifics of encoding the graph structure within the prompt. The GoT architecture enables the user to implement use case specific graph encodings by providing full access to the graph structure.

Parser The Parser extracts information from LLM thoughts. For each such thought, the Parser constructs the *thought state*, which contains this extracted information. The thought state is then used to update the GRS accordingly.

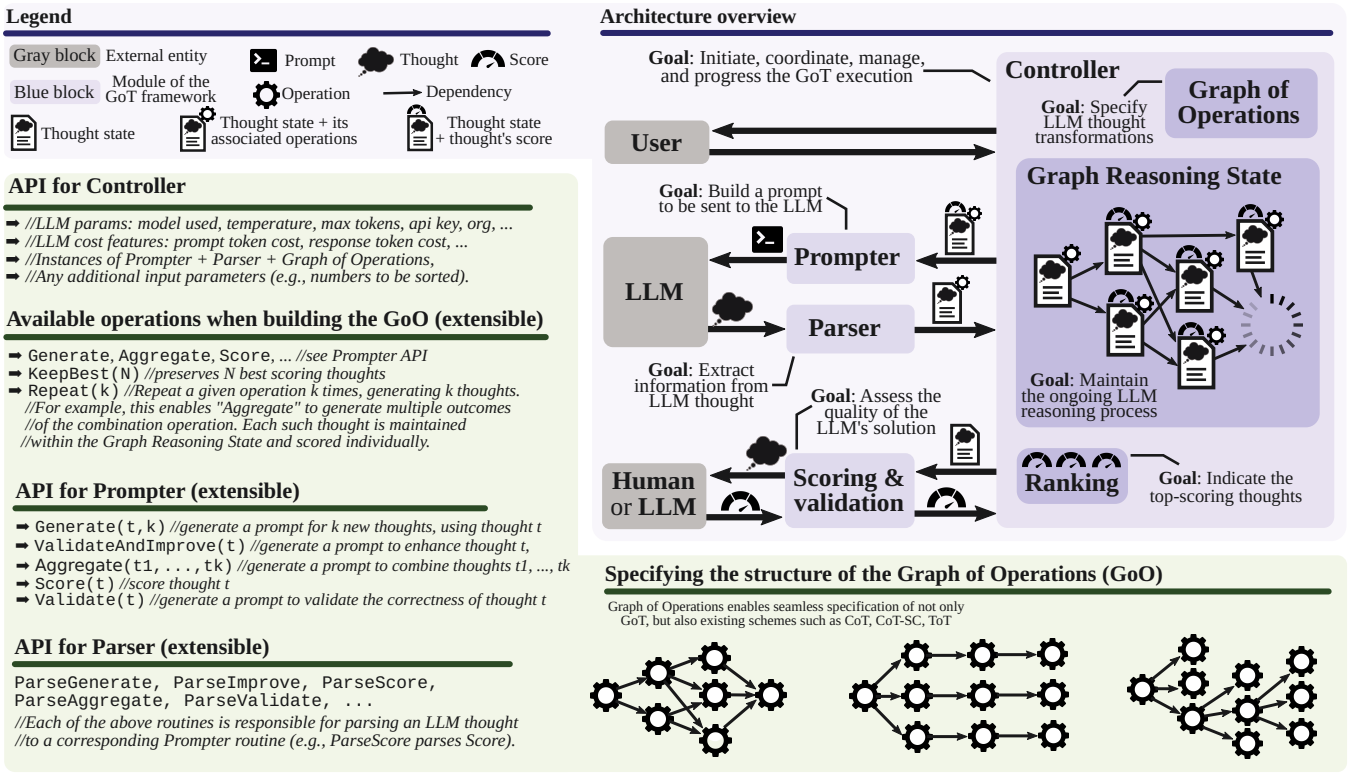

Figure 2: The system architecture of GoT, and the APIs of respective modules. The user can straightforwardly extend the design towards new prompting schemes, experiment with novel thought transformations, and plug in different LLMs. The blue part of the figure contains the architecture overview, and the green part lists the API.

Scoring & Validation Here, we verify whether a given LLM thought satisfies potential correctness conditions, and then we assign it a score. Depending on how the score is derived, the module may consult the LLM. Moreover, depending on the use case, the score may also be assigned by a human. Finally, use cases such as sorting use simple local scoring functions.

Controller The Controller implements a specific strategy for selecting thoughts from its GRS structure. It also selects what transformations should be applied to which thoughts, and then passes this information to the Prompter. It also decides whether the whole process should be finalized, or whether the next round of interaction with the LLM should be initiated. In our current design, this is dictated by the execution plan specified in the GoO.

GoO & GRS The user constructs a GoO instance, which prescribes the execution plan of thought operations. The GoO is a static structure that is constructed once, before the execution starts. Each operation object knows its predecessor and successor operations. Then, during the execution, an instance of the GRS maintains the continually updated information about the LLM reasoning process. This includes which operation has been executed so far, the states of all the generated LLM thoughts, their validity and scores, and any other relevant information.

The above elements offer extensible **APIs**, enabling straightforward implementations of different prompting schemes. The APIs are outlines in the green part of Figure 2, and detailed in the documentation.

4 Example Use Cases

Due to space constraints, we detail one use case (**sorting**). We focus on its decomposition and Graph of Operations, which are central for implementing and executing any workload within GoT. We consider sorting numbers 0–9 with duplicates. The considered LLMs are unable to sort a sequence of such numbers correctly beyond a certain length consistently because duplicate counts do not match.

In GoT, we employ merge-based sorting: First, one decomposes the input sequence of numbers into subarrays. Then, one sorts these subarrays individually, and then respectively merges them into a final solution. Figure 3 illustrates this use case together with its graph decomposition. Here, an LLM thought is a sequence of sorted numbers.

Moreover, we also consider **set operations**, focusing on set intersection. They have numerous applications (particularly set intersection) in problems ranging from genome or document comparisons to pattern matching (Besta et al. 2020, 2021a). Set intersection of two sets is implemented similarly as the sorting. The second input set is split into subsets and the intersection of those subsets with the first input set is determined with the help of the LLM. Afterwards the resulting intersection sets are aggregated for the final re-

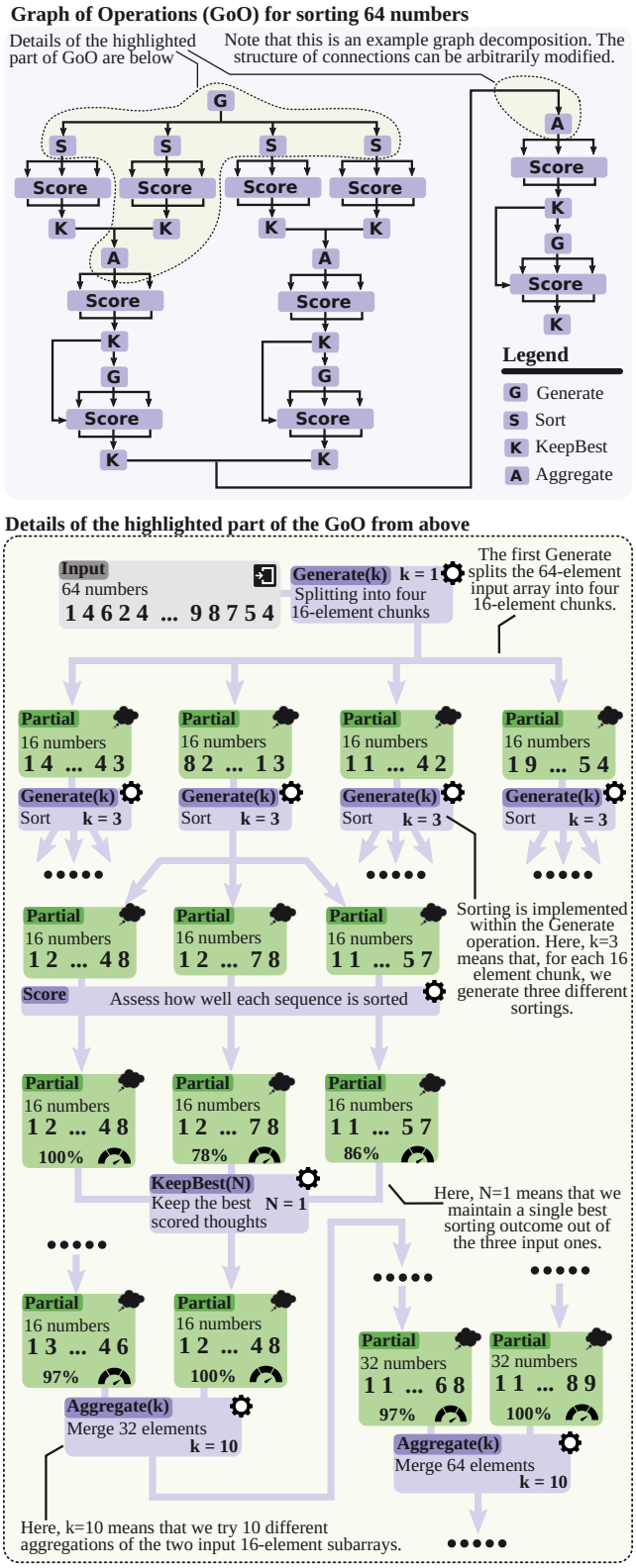

Figure 3: An example graph decomposition of the sorting use case in GoT. All used operations (Generate, Aggregate, Score, KeepBest) are described in Figure 2.

sults. For the evaluation we use different set sizes of 32, 64 and 128 elements and we vary the number of elements found in both sets to be between 25% and 75%.

Keyword counting finds the frequency of keywords in a given category (countries in our example implementation) within the input text. GoT splits the input text into multiple passages, counts the keywords in each passage and aggregates the subresults. The number of passages is configurable and can also be left to the LLM, making it possible to treat each sentence as a separate passage. Here, to score a thought, we first – for each keyword – derive the absolute difference between the computed count and the correct one. We then sum all these differences to get the final score.

Finally, we also provide **document merging**. Here, the goal is to generate a new Non-Disclosure Agreement (NDA) document based on several input ones that partially overlap in terms of their contents. The goal is to ensure minimal amount of duplication, while maximizing information retention. Document merging is broadly applicable in, e.g., legal procedures, where multiple sources of information have to be combined into a single document or article. To score a solution, we query the LLM for two values (3 times for each value, and take the average). The first value corresponds to the solution redundancy (10 indicates no redundancy, 0 implies at least half the information is redundant), the second value stands for information retention (10 indicates all information is retained, 0 says that none is retained). We compute the harmonic mean of these values.

5 The Latency-Volume Tradeoff

We now show that GoT improves upon previous prompting schemes in terms of the tradeoff between latency (number of hops in the graph of thoughts to reach a given final thought) and *volume*. We define volume – for a given thought t – as *the number of preceding LLM thoughts that could have impacted* t. Formally, the volume of t is the number of thoughts from which there exists a path to t in the graph of thoughts. We assume that outputting a single thought costs $O(1)$ time and fix the total cost to $\Theta(n)$ for each prompting scheme.

The structure of the schemes is as follows. CoT-SC consists of k independent chains originating from a single starting thought. ToT is a complete k-ary tree. Finally, in GoT, a complete k-ary tree is joined at its leaves with a "mirrored" k-ary tree of the same size but with its edges reversed.

The analysis is detailed in Table 2. CoT offers a large volume of up to N, but at the cost of a high latency of N. CoT-

Scheme	Latency	Volume
CoT	N	N
CoT-SC	N/k	N/k
ToT	$\log_k N$	$O(\log_k N)$
GoT	$\log_k N$	N

Table 2: Comparison of prompting schemes, with respect to their fundamental tradeoff between latency and volume. *GoT offers the best tradeoff.*

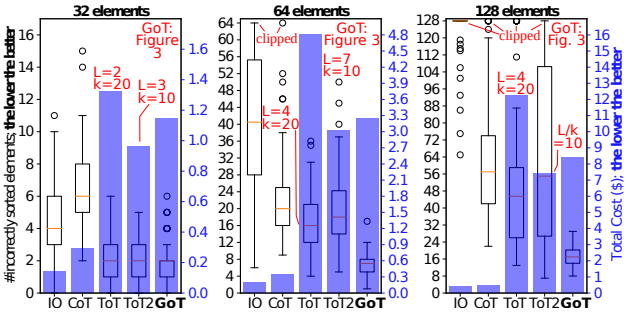

Figure 4: Accuracy and cost in sorting tasks with ChatGPT-3.5. L and k indicate the structure of ToT (see Section 2.2).

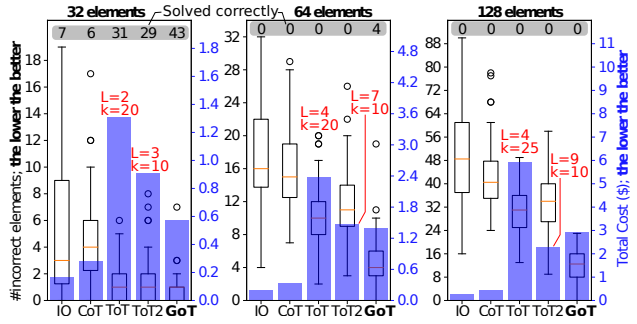

Figure 5: Accuracy and cost in set intersection tasks with ChatGPT-3.5. L and k indicate the structure of ToT (see Sections 2.2 and 5).

SC reduces the latency by a factor of k (which corresponds to its branching factor), but it simultaneously decreases the volume by k as well. ToT offers a latency of $\log_k N$ but also has low volume. GoT is the only scheme to come with both a low latency of $\log_k N$ and a high volume N. This is enabled by the fact that GoT harnesses aggregations of thoughts, making it possible to reach the final thought from any other intermediate thought in the graph decomposition.

6 Evaluation

We show the advantages of GoT over the state of the art. We focus on comparing GoT to ToT, as it was shown to consistently outperform other schemes. Still, for a broad comparison, we also experiment with IO, CoT, and CoT-SC. As our analysis results in a large evaluation space, we present representative results and omit data that does not bring relevant insights (e.g., CoT-SC).

6.1 Evaluation Methodology

We use 100 input samples for each task and comparison baseline. We set the temperature to 1.0 and use a 4k context size unless stated otherwise. For each experiment, we fix the numbers of thoughts in respective schemes to achieve similar costs in each experiment.

Parameters We experiment extensively with the branching factor k and the number of levels L to ensure that we compare GoT to cost-effective and advantageous configurations. We plot two variants of ToT: one with higher k and lower depth (ToT), the other with lower k but higher L (ToT2). We usually aim to achieve a sweet spot in the trade-off between sparser generation rounds (lower k) vs. more rounds (larger L). Usually more responses per round is more expensive (e.g., 80 vs. 60 total responses for Figure 6 but $6 vs. $3 costs). We also try different problem sizes P (e.g., in sorting, P states how many numbers are to be sorted).

Used LLMs Due to budget restrictions, we focus on GPT-3.5. We also experimented with Llama 2, but it was usually worse than GPT-3.5 and also much slower to run, making it infeasible to obtain enough samples.

6.2 Analysis of GoT's Advantages

The results of analysis are in Figure 4 (sorting), 5 (set intersection), and 6 (keyword counting and document merging); see Section 4 for the description of specific use cases. *Overall, GoT improves the quality of outcomes over all the considered baselines and it reduces inference costs compared to ToT.*

GoT vs. ToT GoT improves upon ToT and ToT2 by a large margin over all the considered problem instances. ToT usually comes with somewhat higher quality than ToT2, but simultaneously much higher costs. GoT's costs are always lower than ToT, and comparable (in some cases lower, in others higher) to ToT2. For example, it reduces median error by ≈62%, thereby achieving a higher quality of sorting, for $P = 128$ in comparison to ToT while ensuring >31% cost reductions. These advantages are due to GoT's ability to decompose complex tasks into simpler subtasks, solve these subtasks independently, and then incrementally merge these outcomes into the final result.

GoT vs. IO and CoT GoT consistently delivers much higher quality of outcomes than IO/CoT. For example, for sorting ($P = 64$), GoT's median error is ≈65% and ≈83% lower than, respectively, CoT and IO. Yet, the costs of GoT – and ToT – are much higher than in IO and CoT. This is mostly due to our configuration of CoT, where we do not artificially inflate the lengths of the chains of reasoning if this does not improve the outcomes. The higher costs of GoT and ToT are driven by k new thoughts built for each `Generate` operation; these multiple thoughts are one of the reasons for GoT's superiority in quality.

Increasing Complexity of Tackled Problems Most importantly, the advantages of GoT in the quality *increase for all the baselines with the growing size of the problem* P. For example, in sorting, while for $P = 32$ GoT only negligibly improves upon ToT2, its median error count becomes lower by ≈61% for $P = 64$ and ≈69% for $P = 128$. The quartiles also become respectively better. The results for other schemes also follow the intuition; for example, IO becomes consistently worse with the increasing P, which is expected as a single thought is unlikely to solve a large problem instance. *Overall, this analysis illustrates that GoT is indeed well-suited for elaborate problem cases*, as the execution schedules usually become more complex with the growing problem sizes.

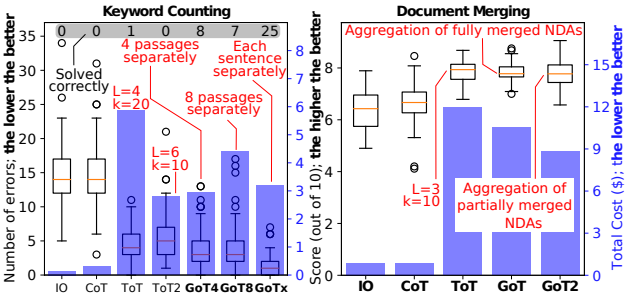

Figure 6: Number of errors and cost in keyword counting and score in document merging with ChatGPT-3.5. L and k indicate the structure of ToT (see Sections 2.2 and 5). Number of samples for document merging: 50; context size for document merging: 16k tokens.

6.3 Discussion on Task Decomposition

When splitting a task into subtasks and then solving these subtasks, the size of responses and the input (in tokens) are reduced proportionally to the degree of the task decomposition. However, the "static" part of the prompt (i.e., few-shot examples) may become a significant overhead (see GoT4 to GoT8 in Figure 6). Here, we observe that these few-shot examples can usually also be reduced in size (e.g., the passages used to demonstrate keyword counting can also be made smaller and still be indicative of the actual input size), thus actively working towards decreasing the cost (e.g., see the difference between GoT8 and GoTx in Figure 6).

The overall goal when conducting graph decomposition is to break down a task to the point, where the LLM can solve it correctly for the majority of time using a single prompt (or with a few additional improvement steps). This significantly lowers the number of improvement/refinement steps needed during the later stages of the graph exploration. Furthermore, as indicated by our results, combining or concatenating subresults is usually an easier task than solving large task instances from scratch. Hence, the LLM is often successful when aggregating the final solution.

7 Related Work

We summarize relations between GoT and related work.

Prompting Paradigms & Approaches We detail different prompting paradigms in Section 1 and Table 1. There are numerous other works related to prompting, including Plan-and-Solve (Wang et al. 2023a), a scheme by Fu et al. (Fu et al. 2022), the self-taught reasoner (Zelikman et al. 2022), a scheme by Shum et al. (Shum, Diao, and Zhang 2023), automatic prompt generation (Shin et al. 2020; Li and Liang 2021; Lester, Al-Rfou, and Constant 2021), concurrent expansion of brief answers in the form of bullet points (Ning et al. 2023), or selecting the best prompt out of a candidate set (Zhou et al. 2022). Most of these schemes could be expressed by the GoT abstraction.

Prompt Chaining In prompt chaining, one cascades different LLMs (Creswell, Shanahan, and Higgins 2022; Nye et al. 2021; Wu, Terry, and Cai 2022; Dohan et al. 2022; Qiao et al. 2023; Wu et al. 2022). One could easily extend GoT so that it can serve as the execution engine for these schemes.

Self-Reflection & Self-Evaluation Self-reflection and self-evaluation were introduced recently (Shinn et al. 2023; Paul et al. 2023; Madaan et al. 2023; Xie et al. 2023; Zhu et al. 2023). In GoT, we partially rely on self-evaluation when expanding the graph of thoughts within a prompt.

LLMs & Planning There are many works on how to plan complex tasks with LLMs (Huang et al. 2022a,b; Zhang et al. 2023; Yao et al. 2023b; Yang et al. 2023; Wang et al. 2023c). GoT could be seen as a generic framework that could potentially be used to enhance such schemes, by offering a paradigm for generating complex graph-based plans.

Graphs & Graph Computing Graphs have become an immensely popular and important part of the general computing landscape (Lumsdaine et al. 2007; Malewicz et al. 2010; Gregor and Lumsdaine 2005a,b; Sakr et al. 2021). Recently, there has been a growing interest in domains such as graph databases (Robinson et al. 2015; Besta et al. 2022b, 2023b,d,c), graph pattern matching (Fan et al. 2010; Cheng et al. 2008; Teixeira et al. 2015; Besta et al. 2021a,b, 2022d), graph streaming (Feng, Meng, and Ammar 2015; Dhulipala, Blelloch, and Shun 2019; Besta et al. 2023a), and graph machine learning as well as graph neural networks (Hamilton, Ying, and Leskovec 2017; Wu et al. 2021; Zhou et al. 2020; Zhang, Cui, and Zhu 2022; Chami et al. 2020; Bronstein et al. 2017; Besta et al. 2022a,c; Gianinazzi et al. 2021; Scarselli et al. 2008). In this work, we harness the graph abstraction as a key mechanism that enhances prompting capabilities in LLMs.

8 Conclusion

Prompt engineering is one of the central new domains of the large language model (LLM) research. It enables using LLMs efficiently, without any model updates. However, designing effective prompts is a challenging task.

In this work, we propose Graph of Thoughts (GoT), a new paradigm that enables the LLM to solve different tasks effectively without any model updates. The key idea is to model the LLM reasoning as an arbitrary graph, where thoughts are vertices and dependencies between thoughts are edges. This enables novel transformations of thoughts, such as aggregation. Human's task solving is often non-linear, and it involves combining intermediate solutions into final ones, or changing the flow of reasoning upon discovering new insights. GoT reflects this with its graph structure.

GoT outperforms other prompting schemes, for example ensuring 62% increase in the quality of sorting over ToT, while simultaneously reducing costs by >31%. We also propose a novel metric for a prompting scheme, the volume of a thought, to indicate the scope of information that a given LLM output could carry with it, where GoT also excels. This provides a step towards more principled prompt engineering.

The graph abstraction has been the foundation of several successful designs in computing and AI over last decades, for example AlphaFold for protein predictions. Our work harnesses it within the realm of prompt engineering.

Acknowledgements

We thank Hussein Harake, Colin McMurtrie, Mark Klein, Angelo Mangili, and the whole CSCS team granting access to the Ault and Daint machines, and for their excellent technical support. We thank Timo Schneider for help with infrastructure at SPCL. This project received funding from the European Research Council (Project PSAP, No. 101002047), and the European High-Performance Computing Joint Undertaking (JU) under grant agreement No. 955513 (MAELSTROM). This project was supported by the ETH Future Computing Laboratory (EFCL), financed by a donation from Huawei Technologies. This project received funding from the European Union's HE research and innovation programme under the grant agreement No. 101070141 (Project GLACIATION).

References

Besta, M.; Fischer, M.; Kalavri, V.; Kapralov, M.; and Hoefler, T. 2023a. Practice of Streaming Processing of Dynamic Graphs: Concepts, Models, and Systems. *IEEE Transactions on Parallel and Distributed Systems*, 34(6): 1860–1876.

Besta, M.; Gerstenberger, R.; Blach, N.; Fischer, M.; and Hoefler, T. 2023b. GDI: A Graph Database Interface Standard. https://github.com/spcl/GDI-RMA. Accessed: 2023-09-05.

Besta, M.; Gerstenberger, R.; Peter, E.; et al. 2023c. Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries. *ACM Comput. Surv.*, 56(2).

Besta, M.; Grob, R.; Miglioli, C.; et al. 2022a. Motif Prediction with Graph Neural Networks. In *28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, 35–45.

Besta, M.; Iff, P.; Scheidl, F.; Osawa, K.; Dryden, N.; Podstawski, M.; Chen, T.; and Hoefler, T. 2022b. Neural Graph Databases. In *First Learning on Graphs Conference*, volume 198 of *Proceedings of Machine Learning Research*, 31:1–31:38. PMLR.

Besta, M.; Kanakagiri, R.; Kwaśniewski, G.; et al. 2021a. SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems. In *54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, 282–297.

Besta, M.; et al. 2020. Communication-Efficient Jaccard Similarity for High-Performance Distributed Genome Comparisons. In *IEEE International Parallel and Distributed Processing Symposium*, IPDPS '20, 1122–1132.

Besta, M.; et al. 2021b. GraphMineSuite: Enabling High-Performance and Programmable Graph Mining Algorithms with Set Algebra. *Proc. VLDB Endow.*, 14(11): 1922–1935.

Besta, M.; et al. 2022c. Parallel and Distributed Graph Neural Networks: An In-Depth Concurrency Analysis. arXiv:2205.09702.

Besta, M.; et al. 2022d. ProbGraph: High-Performance and High-Accuracy Graph Mining with Probabilistic Set Representations. In *International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '22. IEEE.

Besta, M.; et al. 2023d. The Graph Database Interface: Scaling Online Transactional and Analytical Graph Workloads to Hundreds of Thousands of Cores. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23. ACM.

Bronstein, M. M.; Bruna, J.; LeCun, Y.; Szlam, A.; and Vandergheynst, P. 2017. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4): 18–42.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; et al. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems (NeurIPS '20)*, volume 33, 1877–1901. Curran Associates.

Bubeck, S.; et al. 2023. Sparks of Artificial General Intelligence: Early experiments with GPT-4. arXiv:2303.12712.

Chami, I.; et al. 2020. Machine Learning on Graphs: A Model and Comprehensive Taxonomy. arXiv:2005.03675.

Cheng, J.; Yu, J. X.; Ding, B.; Philip, S. Y.; and Wang, H. 2008. Fast Graph Pattern Matching. In *24th International Conference on Data Engineering*, ICDE '08, 913–922. IEEE.

Chowdhery, A.; et al. 2022. PaLM: Scaling Language Modeling with Pathways. arXiv:2204.02311.

Creswell, A.; Shanahan, M.; and Higgins, I. 2022. Selection-Inference: Exploiting Large Language Models for Interpretable Logical Reasoning. arXiv:2205.09712.

Dhulipala, L.; Blelloch, G. E.; and Shun, J. 2019. Low-Latency Graph Streaming Using Compressed Purely-Functional Trees. In *40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '19, 918–934.

Dohan, D.; Xu, W.; Lewkowycz, A.; Austin, J.; Bieber, D.; et al. 2022. Language Model Cascades. In *Beyond Bayes: Paths Towards Universal Reasoning Systems*, Workshop at ICML '22.

Fan, W.; Li, J.; Ma, S.; Tang, N.; Wu, Y.; and Wu, Y. 2010. Graph Pattern Matching: From Intractable to Polynomial Time. *Proc. VLDB Endow.*, 3(1–2): 264–275.

Feng, G.; Meng, X.; and Ammar, K. 2015. DISTINGER: A distributed graph data structure for massive dynamic graph processing. In *Proccedings of the IEEE International Conference on Big Data*, Big Data '15, 1814–1822.

Friston, K. 2008. Hierarchical Models in the Brain. *PLOS Computational Biology*, 4(11): 1–24.

Fu, Y.; et al. 2022. Complexity-Based Prompting for Multi-Step Reasoning. arXiv:2210.00720.

Gianinazzi, L.; Fries, M.; Dryden, N.; et al. 2021. Learning Combinatorial Node Labeling Algorithms. arXiv:2106.03594.

Gregor, D.; and Lumsdaine, A. 2005a. Lifting Sequential Graph Algorithms for Distributed-Memory Parallel Computation. *SIGPLAN Not.*, 40(10): 423–437.

Gregor, D.; and Lumsdaine, A. 2005b. The Parallel BGL: A generic library for distributed graph computations. *Parallel Object-Oriented Scientific Computing (POOSC)*.

Hamilton, W. L.; Ying, R.; and Leskovec, J. 2017. Representation Learning on Graphs: Methods and Applications. *Bulletin of the Technical Committee on Data Engineering*, 40(3): 52–74.

Huang, W.; Abbeel, P.; Pathak, D.; and Mordatch, I. 2022a. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. In *39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, 9118–9147. PMLR.

Huang, W.; Xia, F.; Xiao, T.; Chan, H.; Liang, J.; et al. 2022b. Inner Monologue: Embodied Reasoning through Planning with Language Models. arXiv:2207.05608.

Lester, B.; Al-Rfou, R.; and Constant, N. 2021. The Power of Scale for Parameter-Efficient Prompt Tuning. In *Conference on Empirical Methods in Natural Language Processing*, EMNLP '21, 3045–3059. Association for Computational Linguistics.

Li, X. L.; and Liang, P. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. arXiv:2101.00190.

Long, J. 2023. Large Language Model Guided Tree-of-Thought. arXiv:2305.08291.

Lumsdaine, A.; Gregor, D.; Hendrickson, B.; and Berry, J. 2007. Challenges in Parallel Graph Processing. *Parallel Processing Letters*, 17(1): 5–20.

Madaan, A.; et al. 2023. Self-Refine: Iterative Refinement with Self-Feedback. arXiv:2303.17651.

Malewicz, G.; Austern, M. H.; Bik, A. J.; et al. 2010. Pregel: A System for Large-Scale Graph Processing. In *International Conference on Management of Data*, SIGMOD '10, 135–146. ACM.

Ning, X.; et al. 2023. Skeleton-of-Thought: Large Language Models Can Do Parallel Decoding. arXiv:2307.15337.

Nye, M.; Andreassen, A. J.; Gur-Ari, G.; Michalewski, H.; Austin, J.; Bieber, D.; et al. 2021. Show Your Work: Scratchpads for Intermediate Computation with Language Models. arXiv:2112.00114.

Paul, D.; et al. 2023. REFINER: Reasoning Feedback on Intermediate Representations. arXiv:2304.01904.

Qiao, S.; Ou, Y.; Zhang, N.; Chen, X.; Yao, Y.; et al. 2023. Reasoning with Language Model Prompting: A Survey. In *61st Annual Meeting of the Association for Computational Linguistics*, ACL '23, 5368–5393. Association for Computational Linguistics.

qrdlgit. 2023. graph-of-thoughts Repository. https://github.com/qrdlgit/graph-of-thoughts. Accessed: 2023-10-11.

Radford, A.; et al. 2018. Improving Language Understanding by Generative Pre-Training. https://openai.com/research/language-unsupervised. Accessed: 2023-09-06.

Radford, A.; et al. 2019. Language Models are Unsupervised Multitask Learners. https://openai.com/research/better-language-models. Accessed: 2023-09-06.

Robinson, I.; et al. 2015. *Graph Databases: New Opportunities for Connected Data*. O'Reilly Media, 2nd edition.

Sakr, S.; Bonifati, A.; Voigt, H.; Iosup, A.; Ammar, K.; Angles, R.; et al. 2021. The Future is Big Graphs: A Community View on Graph Processing Systems. *Commun. ACM*, 64(9): 62–71.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2008. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1): 61–80.

Shin, T.; Razeghi, Y.; Logan IV, R. L.; Wallace, E.; and Singh, S. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. arXiv:2010.15980.

Shinn, N.; et al. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv:2303.11366.

Shum, K.; Diao, S.; and Zhang, T. 2023. Automatic Prompt Augmentation and Selection with Chain-of-Thought from Labeled Data. arXiv:2302.12822.

Teixeira, C. H. C.; et al. 2015. Arabesque: A System for Distributed Graph Mining. In *25th Symposium on Operating Systems Principles*, SOSP '15, 425–440. ACM.

Touvron, H.; et al. 2023a. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288.

Touvron, H.; et al. 2023b. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; et al. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems (NIPS '17)*, volume 30. Curran Associates.

Wang, L.; Xu, W.; Lan, Y.; Hu, Z.; Lan, Y.; Lee, R. K.-W.; and Lim, E.-P. 2023a. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. In *61st Annual Meeting of the Association for Computational Linguistics*, ACL '23, 2609–2634. Association for Computational Linguistics.

Wang, X.; et al. 2023b. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *Eleventh International Conference on Learning Representations*, ICLR '23.

Wang, Z.; et al. 2023c. Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents. In *Advances in Neural Information Processing Systems (NeurIPS '23)*, volume 36. Curran Associates.

Wei, J.; et al. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903.

Wu, T.; Jiang, E.; Donsbach, A.; Gray, J.; Molina, A.; Terry, M.; and Cai, C. J. 2022. PromptChainer: Chaining Large Language Model Prompts through Visual Programming. In *Extended Abstracts of the Conference on Human Factors in Computing Systems*, CHI EA '22. ACM.

Wu, T.; Terry, M.; and Cai, C. J. 2022. AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts. In *Conference on Human Factors in Computing Systems*, CHI '22. ACM.

Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; and Yu, P. S. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1): 4–24.

Xie, Y.; et al. 2023. Self-Evaluation Guided Beam Search for Reasoning. In *Advances in Neural Information Processing Systems (NeurIPS '23)*, volume 36. Curran Associates.

Yang, S.; Nachum, O.; Du, Y.; Wei, J.; Abbeel, P.; and Schuurmans, D. 2023. Foundation Models for Decision Making: Problems, Methods, and Opportunities. arXiv:2303.04129.

Yao, S.; Yu, D.; Zhao, J.; Shafran, I.; Griffiths, T. L.; Cao, Y.; and Narasimhan, K. R. 2023a. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Advances in Neural Information Processing Systems (NeurIPS '23)*, volume 36. Curran Associates.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; et al. 2023b. ReAct: Synergizing Reasoning and Acting in Language Models. In *Eleventh International Conference on Learning Representations*, ICLR '23.

Yao, Y.; Li, Z.; and Zhao, H. 2023. Beyond Chain-of-Thought, Effective Graph-of-Thought Reasoning in Large Language Models. arXiv:2305.16582.

Zelikman, E.; et al. 2022. STaR: Bootstrapping Reasoning With Reasoning. In *Advances in Neural Information Processing Systems (NeurIPS '22)*, volume 35, 15476–15488. Curran Associates.

Zhang, S.; Chen, Z.; Shen, Y.; Ding, M.; et al. 2023. Planning with Large Language Models for Code Generation. In *Eleventh International Conference on Learning Representations*, ICLR '23.

Zhang, Z.; Cui, P.; and Zhu, W. 2022. Deep Learning on Graphs: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 34(1): 249–270.

Zhou, J.; Cui, G.; Hu, S.; et al. 2020. Graph neural networks: A review of methods and applications. *AI Open*, 1: 57–81.

Zhou, Y.; et al. 2022. Large Language Models Are Human-Level Prompt Engineers. arXiv:2211.01910.

Zhu, X.; Wang, J.; Zhang, L.; Zhang, Y.; Huang, Y.; Gan, R.; Zhang, J.; and Yang, Y. 2023. Solving Math Word Problems via Cooperative Reasoning induced Language Models. In *61st Annual Meeting of the Association for Computational Linguistics*, ACL '23, 4471–4485. Association for Computational Linguistics.