arXiv:2308.09687v4 [cs.CL] 6 Feb 2024

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.

Website & code: https://github.com/spcl/graph-of-thoughts

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 [65], such as GPT [13, 14, 53, 54], PaLM [19], or LLaMA [63].

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) [71] 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) [67], 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) [43, 75, 77], 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 [28]. 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), 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 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

---

[*] Equal contribution

many other questions, we carefully design a modular architecture 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 [64].

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[1] 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.

Scheme	**Sc?**	**Mc?**	**Tr?**	**Ag?**
Chain-of-Thought (CoT) [71]	🔋	✖	✖	✖
Self-Consistency with CoT [67]	🔋	🔋	✖	✖
Thought decomposition [75]	🔋	🔋	🪫	✖
Tree-of-Thought (ToT) [43]	🔋	🔋	🔋	✖
Tree of Thoughts (ToT) [77]	🔋	🔋	🔋	✖
Graph of Thoughts (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.

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 Background & Notation

We first outline background concepts and notation.

2.1 Language Models & In-Context Learning

The **conversation with the LLM** consists of user messages (*prompts*) and LLM replies (*thoughts*). We follow the established notation [77] 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. We purposefully do not prescribe what is a single "thought", and instead make it use-case specific. Hence, a single thought can be a paragraph (e.g., in article summary), a document (e.g., in document generation), a block of code (e.g., in code debugging or optimization), and so on.

We next describe specific **prompting approaches**.

Input-Output (IO) The Input-Output (IO) prompting is a straightforward approach, in which we use an LLM to turn an input sequence x into the output y *directly*, without any intermediate thoughts.

Chain-of-Thought (CoT) Second, in Chain-of-Thought (CoT), one introduces intermediate thoughts $a_1, a_2, ...$ between x and y. This strategy was shown to significantly enhance various LM tasks over the plain IO baseline, such as mathematical puzzles [71] or general mathematical reasoning [24].

Multiple CoTs Third, one can generalize CoT into *multiple CoTs* by generating *several* (independent) k CoTs, and returning the one with the best output (according to some prescribed scoring metric). It was introduced by Wang et al. in the scheme called Self-Consistency with CoT (CoT-SC) [67]. This approach enhances CoT because it offers an opportunity to explore different reasoning paths. However, it does not offer "local exploration" within a path, such as backtracking.

Tree of Thoughts (ToT) Finally, the Tree of Thoughts (ToT) scheme was introduced independently by Yao [77] and Long [43] (where it is referred to as Tree-of-Thought); it was used implicitly to a certain degree by other schemes such as thought decomposition [75]. It enhances CoT-SC by modeling the process or reasoning as a *tree* of thoughts. A single tree node represents a partial solution. Based on a given node, the *thought generator* constructs a given number k of new nodes. Then, the *state evaluator* generates scores for each such new node. Depending on the use case, the evaluation could be conducted using an LLM itself, or it can harness human scores. Finally, the schedule of extending the tree is dictated by the utilized search algorithm (for example BFS or DFS).

3 The GoT Framework

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

---

[1] Note that we do not include a recent scheme called Graph-of-Thought [79] 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 [52] does not enable general graph-based reasoning and harnesses instead ToT with BFS.

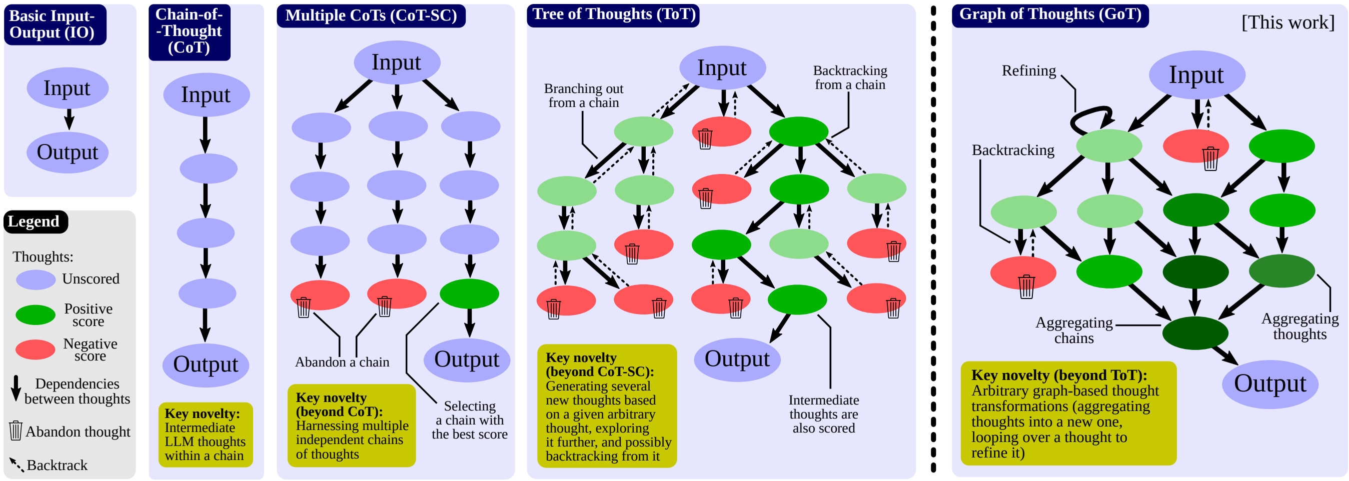

Basic Input-Output (IO)
Input
Output
Legend
Thoughts:
Unscored
Positive score
Negative score
Dependencies between thoughts
Abandon thought
Backtrack
Chain-of--Thought (CoT)
Input
Output
Key novelty: Intermediate LLM thoughts within a chain
Multiple CoTs (CoT-SC)
Input
Output
Abandon a chain
Selecting a chain with the best score
Key novelty (beyond CoT): Harnessing multiple independent chains of thoughts
Tree of Thoughts (ToT)
Input
Branching out from a chain
Backtracking from a chain
Output
Intermediate thoughts are also scored
Key novelty (beyond CoT-SC): Generating several new thoughts based on a given arbitrary thought, exploring it further, and possibly backtracking from it
Graph of Thoughts (GoT)
[This work]
Input
Refining
Backtracking
Aggregating chains
Aggregating thoughts
Output
Key novelty (beyond ToT): Arbitrary graph-based thought transformations (aggregating thoughts into a new one, looping over a thought to refine it)

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

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.

3.1 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. G is directed and thus the edges are a subset of ordered vertex pairs $E \subseteq V \times V$. 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.

In certain use cases, graph nodes belong to different *classes*. For example, in writing tasks, some vertices model *plans of writing a paragraph*, while other vertices model *the actual paragraphs of text*. In such cases, GoT embraces a **heterogeneous graph** $G = (V, E, c)$ to model the LLM reasoning, where c maps vertices V into their respective classes C (in the above case, it would be $C = \{plan, par\}$). Hence, any vertex v can model different aspects of reasoning.

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.

3.2 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. We illustrate examples of aggregation and generation in Figure 2.

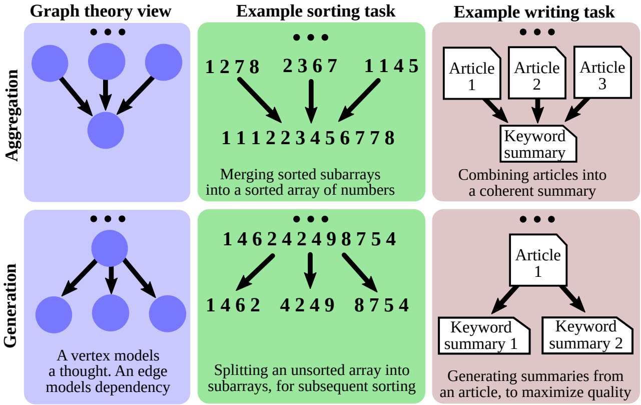

Graph theory view
Example sorting task
Example writing task
Aggregation
1 2 7 8
2 3 6 7
1 1 4 5
1 1 1 2 2 3 4 5 6 7 7 8
Merging sorted subarrays into a sorted array of numbers
Article 1
Article 2
Article 3
Keyword summary
Combining articles into a coherent summary
Generation
A vertex models a thought. An edge models dependency
1 4 6 2 4 2 4 9 8 7 5 4
1 4 6 2
4 2 4 9
8 7 5 4
Splitting an unsorted array into subarrays, for subsequent sorting
Article 1
Keyword summary 1
Keyword summary 2
Generating summaries from an article, to maximize quality

Figure 2: Examples of aggregation and generation thought transformations.

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). Here, it is the user's responsibility to ensure that the sets V^+, E^+, V^-, and E^- come with consistent transformations (i.e., for example, that the user does not attempt to remove a vertex that does not exist). 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.

The specific form of $\mathcal{T}$ and how it impacts G depends on a specific transformation. We first detail the primary graph-enabled thought transformations, and then proceed to describe how GoT embraces the transformations from the earlier schemes. Unless stated otherwise, $V^- = E^- = \emptyset$.

Aggregation Transformations 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.

Refining Transformations 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.

Generation Transformations 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 $E^+ = \{(v, v_1^+), ..., (v, v_k^+)\}$.

3.3 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 5. 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).

4 System Architecture & Extensibility

The GoT architecture consists of a set of interacting modules, see Figure 3 (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).

4.1 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.

4.2 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.

4.3 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.

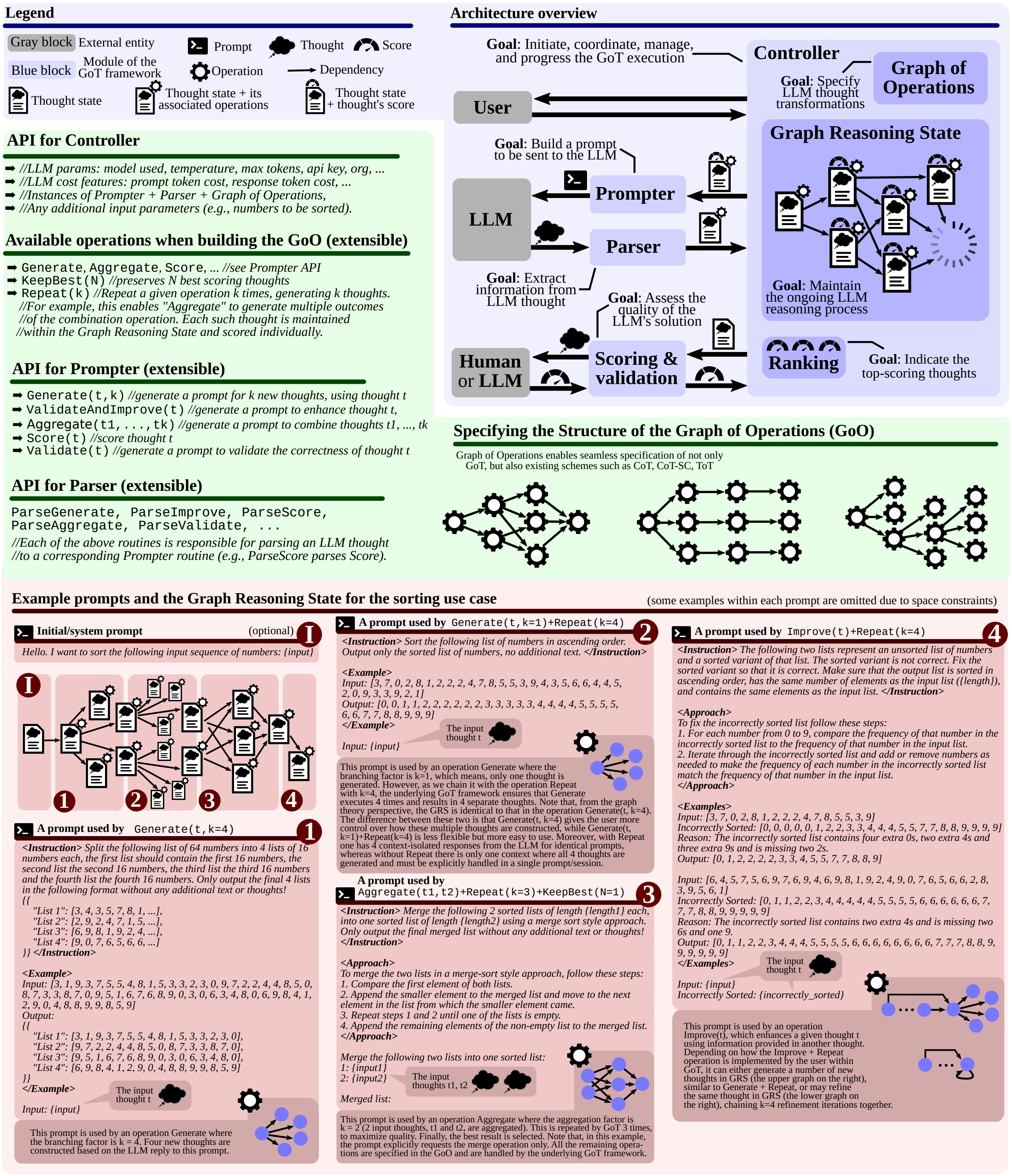
4.4 Controller

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.

4.5 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 3, and detailed in the documentation. We also provide **examples of prompts** used by these operations and a corresponding GRS in the red part of Figure 3.


Legend
Gray block External entity
Blue block Module of the GoT framework
Prompt
Thought
Score
Operation
Dependency
Thought state
Thought state + its associated operations
Thought state + thought's score
Architecture overview
Goal: Initiate, coordinate, manage, and progress the GoT execution
Controller
Goal: Specify LLM thought transformations
Graph of Operations
User
Goal: Build a prompt to be sent to the LLM
Prompter
LLM
Parser
Graph Reasoning State
Goal: Extract information from LLM thought
Goal: Assess the quality of the LLM's solution
Goal: Maintain the ongoing LLM reasoning process
Human or LLM
Scoring & validation
Ranking
Goal: Indicate the top-scoring thoughts
I
1
2
3
4

API for Controller

- *//LLM params: model used, temperature, max tokens, api key, org, ...*
- *//LLM cost features: prompt token cost, response token cost, ...*
- *//Instances of Prompter + Parser + Graph of Operations,*
- *//Any additional input parameters (e.g., numbers to be sorted).*

Available operations when building the GoO (extensible)

- `Generate`, `Aggregate`, `Score`, ... *//see Prompter API*
- `KeepBest(N)` *//preserves N best scoring thoughts*
- `Repeat(k)` *//Repeat a given operation k times, generating k thoughts.*
  //For example, this enables "Aggregate" to generate multiple outcomes
  //of the combination operation. Each such thought is maintained
  //within the Graph Reasoning State and scored individually.

API for Prompter (extensible)

- `Generate(t,k)` *//generate a prompt for k new thoughts, using thought t*
- `ValidateAndImprove(t)` *//generate a prompt to enhance thought t,*
- `Aggregate(t1,...,tk)` *//generate a prompt to combine thoughts t1, ..., tk*
- `Score(t)` *//score thought t*
- `Validate(t)` *//generate a prompt to validate the correctness of thought t*

API for Parser (extensible)

```
ParseGenerate, ParseImprove, ParseScore,
ParseAggregate, ParseValidate, ...
```

//Each of the above routines is responsible for parsing an LLM thought
//to a corresponding Prompter routine (e.g., ParseScore parses Score).

Specifying the Structure of the Graph of Operations (GoO)

Graph of Operations enables seamless specification of not only GoT, but also existing schemes such as CoT, CoT-SC, ToT

Example prompts and the Graph Reasoning State for the sorting use case

(some examples within each prompt are omitted due to space constraints)

Initial/system prompt (optional) I

Hello. I want to sort the following input sequence of numbers: {input}

A prompt used by `Generate(t,k=4)` 1

<Instruction> *Split the following list of 64 numbers into 4 lists of 16 numbers each, the first list should contain the first 16 numbers, the second list the second 16 numbers, the third list the third 16 numbers and the fourth list the fourth 16 numbers. Only output the final 4 lists in the following format without any additional text or thoughts!*
{{
"List 1": [3, 4, 3, 5, 7, 8, 1, ...],
"List 2": [2, 9, 2, 4, 7, 1, 5, ...],
"List 3": [6, 9, 8, 1, 9, 2, 4, ...],
"List 4": [9, 0, 7, 6, 5, 6, 6, ...]
}} ***</Instruction>***

<Example>
Input: [3, 1, 9, 3, 7, 5, 5, 4, 8, 1, 5, 3, 3, 2, 3, 0, 9, 7, 2, 2, 4, 4, 8, 5, 0, 8, 7, 3, 3, 8, 7, 0, 9, 5, 1, 6, 7, 6, 8, 9, 0, 3, 0, 6, 3, 4, 8, 0, 6, 9, 8, 4, 1, 2, 9, 0, 4, 8, 8, 9, 9, 8, 5, 9]
Output:
{{
"List 1": [3, 1, 9, 3, 7, 5, 5, 4, 8, 1, 5, 3, 3, 2, 3, 0],
"List 2": [9, 7, 2, 2, 4, 4, 8, 5, 0, 8, 7, 3, 3, 8, 7, 0],
"List 3": [9, 5, 1, 6, 7, 6, 8, 9, 0, 3, 0, 6, 3, 4, 8, 0],
"List 4": [6, 9, 8, 4, 1, 2, 9, 0, 4, 8, 8, 9, 9, 8, 5, 9]
}}
</Example>

Input: {input}

This prompt is used by an operation Generate where the branching factor is k = 4. Four new thoughts are constructed based on the LLM reply to this prompt.

A prompt used by `Generate(t,k=1)+Repeat(k=4)` 2

<Instruction> *Sort the following list of numbers in ascending order. Output only the sorted list of numbers, no additional text.* ***</Instruction>***

<Example>
Input: [3, 7, 0, 2, 8, 1, 2, 2, 2, 4, 7, 8, 5, 5, 3, 9, 4, 3, 5, 6, 6, 4, 4, 5, 2, 0, 9, 3, 3, 9, 2, 1]
Output: [0, 0, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 9]
</Example>

Input: {input}

This prompt is used by an operation Generate where the branching factor is k=1, which means, only one thought is generated. However, as we chain it with the operation Repeat with k=4, the underlying GoT framework ensures that Generate executes 4 times and results in 4 separate thoughts. Note that, from the graph theory perspective, the GRS is identical to that in the operation Generate(t, k=4). The difference between these two is that Generate(t, k=4) gives the user more control over how these multiple thoughts are constructed, while Generate(t, k=1)+Repeat(k=4) is less flexible but more easy to use. Moreover, with Repeat one has 4 context-isolated responses from the LLM for identical prompts, whereas without Repeat there is only one context where all 4 thoughts are generated and must be explicitly handled in a single prompt/session.

A prompt used by `Aggregate(t1,t2)+Repeat(k=3)+KeepBest(N=1)` 3

<Instruction> *Merge the following 2 sorted lists of length {length1} each, into one sorted list of length {length2} using a merge sort style approach. Only output the final merged list without any additional text or thoughts!*
</Instruction>

<Approach>
To merge the two lists in a merge-sort style approach, follow these steps:
1. Compare the first element of both lists.
2. Append the smaller element to the merged list and move to the next element in the list from which the smaller element came.
3. Repeat steps 1 and 2 until one of the lists is empty.
4. Append the remaining elements of the non-empty list to the merged list.
</Approach>

Merge the following two lists into one sorted list:
1: {input1}
2: {input2}

Merged list:

This prompt is used by an operation Aggregate where the aggregation factor is k = 2 (2 input thoughts, t1 and t2, are aggregated). This is repeated by GoT 3 times, to maximize quality. Finally, the best result is selected. Note that, in this example, the prompt explicitly requests the merge operation only. All the remaining operations are specified in the GoO and are handled by the underlying GoT framework.

A prompt used by `Improve(t)+Repeat(k=4)` 4

<Instruction> *The following two lists represent an unsorted list of numbers and a sorted variant of that list. The sorted variant is not correct. Fix the sorted variant so that it is correct. Make sure that the output list is sorted in ascending order, has the same number of elements as the input list ({length}), and contains the same elements as the input list.* ***</Instruction>***

<Approach>
To fix the incorrectly sorted list follow these steps:
1. For each number from 0 to 9, compare the frequency of that number in the incorrectly sorted list to the frequency of that number in the input list.
2. Iterate through the incorrectly sorted list and add or remove numbers as needed to make the frequency of each number in the incorrectly sorted list match the frequency of that number in the input list.
</Approach>

<Examples>
Input: [3, 7, 0, 2, 8, 1, 2, 2, 2, 4, 7, 8, 5, 5, 3, 9]
Incorrectly Sorted: [0, 0, 0, 0, 0, 1, 2, 2, 3, 3, 4, 4, 4, 5, 5, 7, 7, 8, 8, 9, 9, 9, 9]
Reason: The incorrectly sorted list contains four extra 0s, two extra 4s and three extra 9s and is missing two 2s.
Output: [0, 1, 2, 2, 2, 2, 3, 3, 4, 5, 5, 7, 7, 8, 8, 9]

Input: [6, 4, 5, 7, 5, 6, 9, 7, 6, 9, 4, 6, 9, 8, 1, 9, 2, 4, 9, 0, 7, 6, 5, 6, 6, 2, 8, 3, 9, 5, 6, 1]
Incorrectly Sorted: [0, 1, 1, 2, 2, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 7, 7, 7, 8, 8, 9, 9, 9, 9, 9]
Reason: The incorrectly sorted list contains two extra 4s and is missing two 6s and one 9.
Output: [0, 1, 1, 2, 2, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 8, 8, 9, 9, 9, 9, 9, 9]
</Examples>

Input: {input}
Incorrectly Sorted: {incorrectly_sorted}

This prompt is used by an operation Improve(t), which enhances a given thought t using information provided in another thought. Depending on how the Improve + Repeat operation is implemented by the user within GoT, it can either generate a number of new thoughts in GRS (the upper graph on the right), similar to Generate + Repeat, or may refine the same thought in GRS (the lower graph on the right), chaining k=4 refinement iterations together.

Figure 3: 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, the green part lists the API, and the red part contains example prompts together with a GRS and operations involved.

5 Example Use Cases

We now describe several use cases of GoT. We detail one use case (sorting) and summarize the others.

5.1 Sorting

We focus on the decomposition of the sorting use case 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 4 illustrates this use case together with its graph decomposition. Here, an LLM thought is a sequence of sorted numbers.

To score an outcome, denote an input sequence with $[a_1, a_2, ..., a_n]$ and an output one with $[b_1, b_2, ..., b_m]$. We use the following score that determines "the scope" of errors:

$$\text{error-scope} = X + Y$$

where $p \in \{1, ..., m\}$, $q \in \{1, ..., n\}$, and

$$X = \sum_{i=1}^{m-1} \operatorname{sgn}(\max(b_i - b_{i+1}, 0)),$$

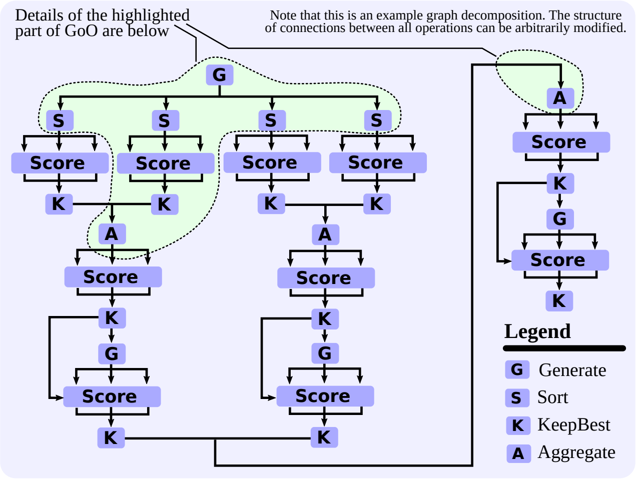
$$Y = \sum_{i=0}^{9} \left|\, |\{b_p : b_p = i\}| - |\{a_q : a_q = i\}| \,\right|$$

Here, X indicates how many consecutive pairs of numbers are incorrectly sorted. If two numbers i and $i + 1$ are incorrectly sorted (i.e., $b_i > b_{i+1}$), then the expression within the summation returns 1, increasing the error score by one. For two numbers correctly sorted, this expression amounts to 0. Then, Y determines how well a given output sequence preserves the frequency of output numbers. Specifically, for each considered number x ($x \in \{0, ..., 9\}$), we obtain the difference between the count of input elements being equal to x, vs. the count of output elements equal to x. For an output sequence perfectly preserving the frequency of x, this would amount to 0. Any single "deviation" in this count, increases the "error scope" by 1. We then sum this over all considered values of x. When plotting this score, to improve the clarity of plots, we additionally apply clipping $\min(\text{error-scope}, n)$, as some baselines (IO, CoT) result in large numbers of outliers with high error scope. Finally, to use a "positive score" describing "the scope of correctly sorted" elements, one can use the value $\max(n - \text{error-scope}, 0)$.

5.2 Set Operations

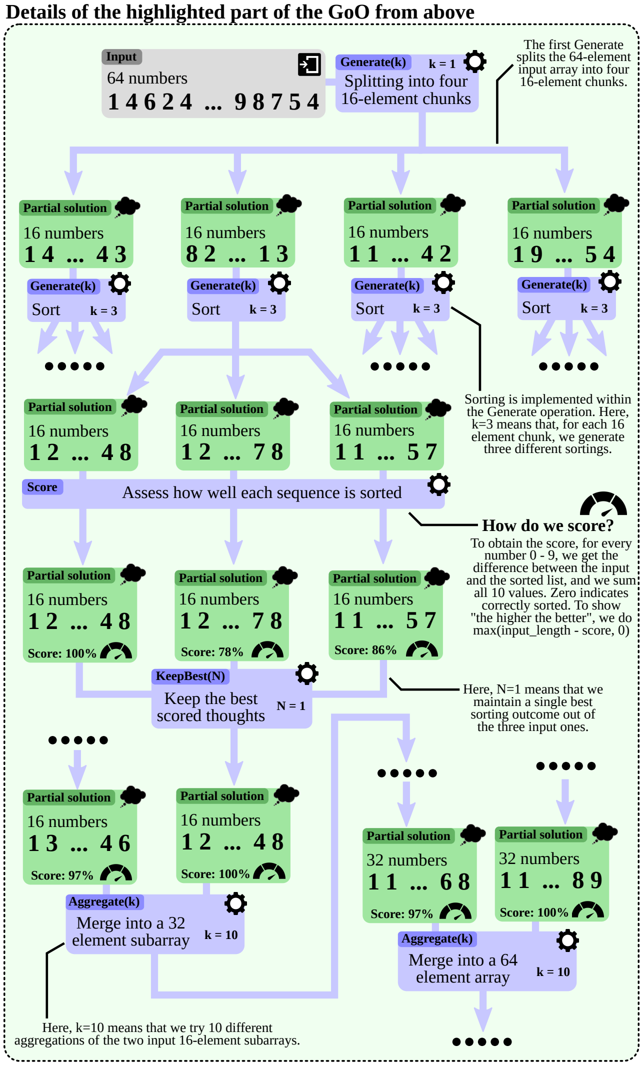
Moreover, we also consider set operations, focusing on set intersection. They have numerous applications (particularly set intersection) in problems ranging from genome or document comparisons to pattern matching [9–11, 20, 27, 38, 50,

Graph of Operations (GoO) for sorting 64 numbers


Details of the highlighted part of GoO are below

Note that this is an example graph decomposition. The structure of connections between all operations can be arbitrarily modified.

Details of the highlighted part of the GoO from above


The first Generate splits the 64-element input array into four 16-element chunks.

Sorting is implemented within the Generate operation. Here, k=3 means that, for each 16 element chunk, we generate three different sortings.

How do we score?
To obtain the score, for every number 0 - 9, we get the difference between the input and the sorted list, and we sum all 10 values. Zero indicates correctly sorted. To show "the higher the better", we do max(input_length - score, 0)

Here, N=1 means that we maintain a single best sorting outcome out of the three input ones.

Here, k=10 means that we try 10 different aggregations of the two input 16-element subarrays.

Figure 4: An example graph decomposition of the sorting use case in GoT. All used operations (Generate, Aggregate, Score, KeepBest) are described in Figure 3.

58]. 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 results. 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%.

Our score indicates the total number of missing or incorrectly included elements in the final intersection. Specifically, denote two input sets with $A = [a_1, a_2, ..., a_n]$ and $B = [b_1, b_2, ..., b_n]$, and the output set with $C = [c_1, c_2, ..., c_m]$. Then,

$$\text{error-scope} = X_1 + X_2 + X_d$$

where $X_1 = |C \setminus (A \cap B)|$ are the number of elements in C that are not supposed to be there, $X_2 = |(A \cap B) \setminus C|$ are the number of elements missing from C, and X_d is the number of duplicates in C (because the LLM expresses the set as a list in natural language). Finally, to use a "positive score" describing "the scope of correctly computed" elements, one can use the value $\max(n - \text{error-scope}, 0)$.

5.3 Keyword Counting

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.

5.4 Document Merging

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.

6 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-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.

Scheme	**Latency**	**Volume**
Chain-of-Thought (CoT)	N	N
Self-Consistency with CoT (CoT-SC)	N/k	N/k
Tree of Thoughts (ToT)	$\log_k N$	$O(\log_k N)$
Graph of Thoughts (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.**

7 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).

7.1 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 tradeoff 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 7 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).

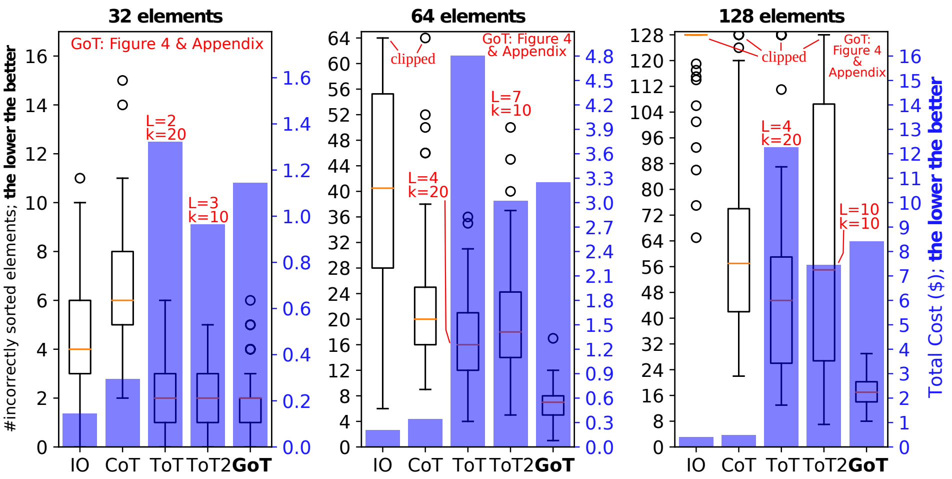


Figure 5: Number of errors and cost in sorting tasks with ChatGPT-3.5. L and k indicate the structure of ToT (see Sections 3.2 and 6).

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.

7.2 Analysis of GoT's Advantages

The results of the analysis are in Figure 5 (sorting), 6 (set intersection), 7 (keyword counting), and 8 (document merging); see Section 5 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.

7.3 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 7). 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 7).

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.

8 Related Work

We summarize relations between GoT and related work.

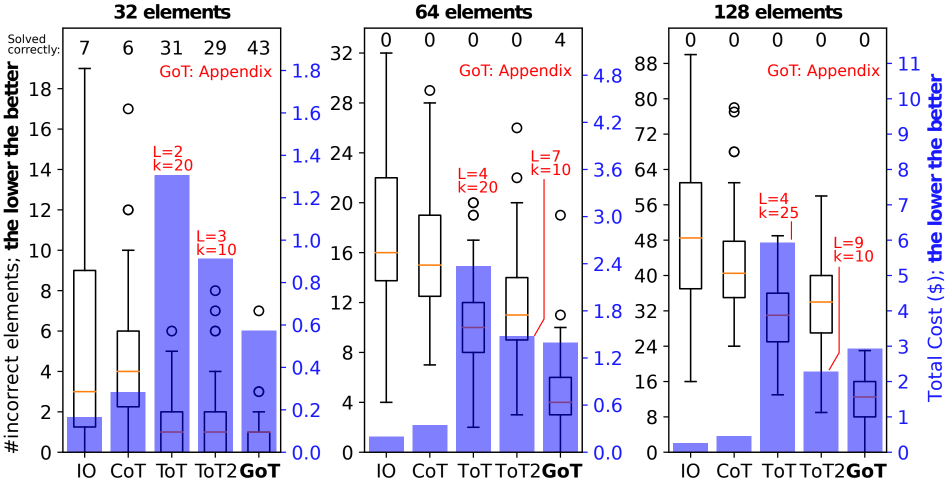


Figure 6: Number of errors and cost in set intersection tasks with ChatGPT-3.5. L and k indicate the structure of ToT (see Sections 3.2 and 6).

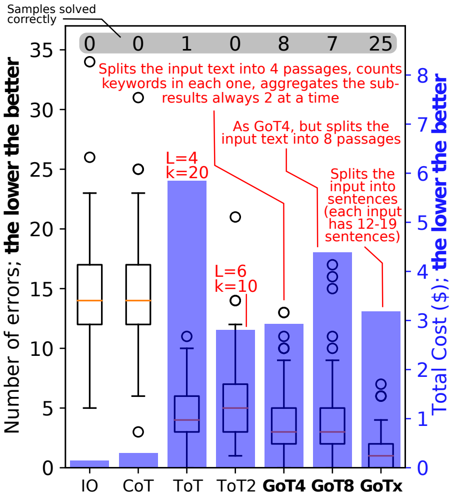


Figure 7: Number of errors and cost in keyword counting with ChatGPT-3.5. L and k indicate the structure of ToT (see Sections 3.2 and 6).

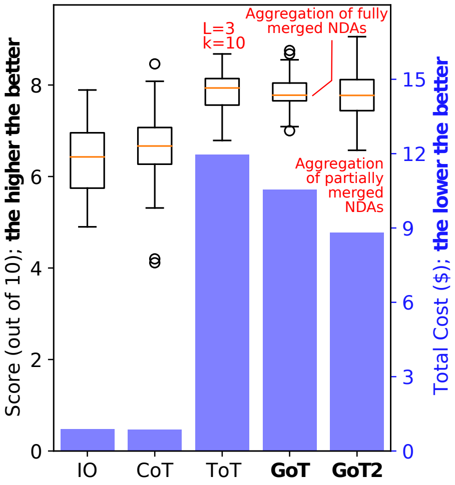


Figure 8: Score and cost in document merging with ChatGPT-3.5. L and k indicate the structure of ToT (see Sections 3.2 and 6). Number of samples: 50; context size: 16k tokens.

8.1 Prompting Paradigms & Approaches

We detail different prompting paradigms in Section 1 and Table 1. There are numerous other works related to prompting. We now briefly summarize selected most related ones; more extensive descriptions can be found in dedicated surveys [34, 40, 69, 70]. Wang et al. proposed Plan-and-Solve, an approach to enhance CoT with an explicit planning stage [66]. Using complexity-based criteria to enhance prompting within a CoT was designed by Fu et al. [29, 67]. The self-taught reasoner (STaR) [80] generates several chain of thoughts, and selects the ones that are valid. Similarly, a scheme by Shum et al. [61] generates a pool of CoT candidates, and selects the best candidate based on whether the candidates match the ground truth and on a policy gradient-based method. Automatic prompt generation overcomes the issues of scaling in CoT [41, 42, 59]. Zhou et al. propose to harness selecting the best prompt out of a candidate set [84]. Skeleon-of-Thought [47] generates at first a number of skeleton answers (brief bullet points of 3 to 5 words) and expands on these points in parallel in a second step.

Finally, in prompt chaining, one cascades different LLMs. This enables prompting different LLMs via different contexts, enabling more powerful reasoning [21, 23, 48, 51, 72, 73, 73]. GoT is orthogonal to this class of schemes, as it focuses on a single context capabilities.

8.2 Self-Reflection & Self-Evaluation

Self-reflection and self-evaluation were introduced recently [45, 49, 60, 75, 85]. They are used to enhance different tasks, for example for code generation [17] or computer operation tasks [39]. In GoT, we partially rely on self-evaluation when taking decisions on how to expand the graph of thoughts within a prompt.

8.3 LLMs & Planning

There are many works recently on how to plan complex tasks with LLMs [36, 37, 68, 76, 78, 81]. 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.

8.4 Graphs and Graph Computing

Graphs have become an immensely popular and important part of the general computing landscape [31, 32, 44, 46, 56]. Recently, there has been a growing interest in domains such as graph databases [2–4, 7, 55], graph pattern matching [8, 10, 11, 18, 25, 62], graph streaming [1, 22, 26], and graph machine learning as well as graph neural networks [5, 6, 12, 16, 30, 33, 33, 57, 74, 82, 83]. The graph abstraction has been fruitful for many modern research domains, such as social sciences (e.g., studying human interactions), bioinformatics (e.g., analyzing protein structures), chemistry (e.g., designing chemical compounds), medicine (e.g., drug discovery), cybersecurity (e.g., identifying intruder machines), healthcare (e.g., exposing groups of people who submit fraudulent claims), web graph analysis (e.g., providing accurate search services), entertainment services (e.g., predicting movie popularity), linguistics (e.g., modeling relationships between words), transportation (e.g., finding efficient routes), physics (e.g., understanding phase transitions and critical phenomena), and many others [15, 20, 35, 38, 44]. In this work, we harness the graph abstraction as a key mechanism that enhances prompting capabilities in LLMs.

9 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

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

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

[3] Besta, M.; Gerstenberger, R.; Fischer, M.; Podstawski, M.; Blach, N.; Egeli, B.; Mitenkov, G.; Chlapek, W.; Michalewicz, M.; Niewiadomski, H.; Müller, J.; and Hoefler, T. 2023. The Graph Database Interface: Scaling Online Transactional and Analytical Graph Workloads to Hundreds of Thousands of Cores. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23. ACM.

[4] Besta, M.; Gerstenberger, R.; Peter, E.; Fischer, M.; Podstawski, M.; Barthels, C.; Alonso, G.; and Hoefler, T. 2023. Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries. *ACM Comput. Surv.*, 56(2).

[5] Besta, M.; Grob, R.; Miglioli, C.; Bernold, N.; Kwaśniewski, G.; Gjini, G.; Kanakagiri, R.; Ashkboos, S.; Gianinazzi, L.; Dryden, N.; and Hoefler, T. 2022. Motif Prediction with Graph Neural Networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '22, 35–45.

[6] Besta, M.; and Hoefler, T. 2022. Parallel and Distributed Graph Neural Networks: An In-Depth Concurrency Analysis. arXiv:2205.09702.

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

[8] Besta, M.; Kanakagiri, R.; Kwaśniewski, G.; Ausavarungnirun, R.; Beránek, J.; Kanellopoulos, K.; Janda, K.; Vonarburg-Shmaria, Z.; Gianinazzi, L.; Stefan, I.; Luna, J. G.; Golinowski, J.; Copik, M.; Kapp-Schwoerer, L.; Di Girolamo, S.; Blach, N.; Konieczny, M.; Mutlu, O.; and Hoefler, T. 2021. SISA: Set-Centric Instruction Set Architecture for Graph Mining on Processing-in-Memory Systems. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, 282–297.

[9] Besta, M.; Kanakagiri, R.; Mustafa, H.; Karasikov, M.; Rätsch, G.; Hoefler, T.; and Solomonik, E. 2020. Communication-Efficient Jaccard Similarity for High-Performance Distributed Genome Comparisons. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, IPDPS '20, 1122–1132.

[10] Besta, M.; Miglioli, C.; Labini, P. S.; Tětek, J.; Iff, P.; Kanakagiri, R.; Ashkboos, S.; Janda, K.; Podstawski, M.; Kwaśniewski, G.; Gleinig, N.; Vella, F.; Mutlu, O.; and Hoefler, T. 2022. ProbGraph: High-Performance and High-Accuracy Graph Mining with Probabilistic Set Representations. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '22. IEEE.

[11] Besta, M.; Vonarburg-Shmaria, Z.; Schaffner, Y.; Schwarz, L.; Kwaśniewski, G.; Gianinazzi, L.; Beranek, J.; Janda, K.; Holenstein, T.; Leisinger, S.; Tatkowski, P.; Ozdemir, E.; Balla, A.; Copik, M.; Lindenberger, P.; Konieczny, M.; Mutlu, O.; and Hoefler, T. 2021. GraphMineSuite: Enabling High-Performance and Programmable Graph Mining Algorithms with Set Algebra. *Proc. VLDB Endow.*, 14(11): 1922–1935.

[12] 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.

[13] Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems (NeurIPS '20)*, volume 33, 1877–1901. Curran Associates.

[14] Bubeck, S.; Chandrasekaran, V.; Eldan, R.; Gehrke, J.; Horvitz, E.; Kamar, E.; Lee, P.; Lee, Y. T.; Li, Y.; Lundberg, S.; Nori, H.; Palangi, H.; Ribeiro, M. T.; and Zhang, Y. 2023. Sparks of Artificial General Intelligence: Early experiments with GPT-4. arXiv:2303.12712.

[15] Chakrabarti, D.; and Faloutsos, C. 2006. Graph Mining: Laws, Generators, and Algorithms. *ACM Comput. Surv.*, 38(1).

[16] Chami, I.; Abu-El-Haija, S.; Perozzi, B.; Ré, C.; and Murphy, K. 2020. Machine Learning on Graphs: A Model and Comprehensive Taxonomy. arXiv:2005.03675.

[17] Chen, X.; Lin, M.; Schärli, N.; and Zhou, D. 2023. Teaching Large Language Models to Self-Debug. arXiv:2304.05128.

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

[19] Chowdhery, A.; Narang, S.; Devlin, J.; Bosma, M.; Mishra, G.; Roberts, A.; Barham, P.; Chung, H. W.; Sutton, C.; Gehrmann, S.; Schuh, P.; Shi, K.; Tsvyashchenko, S.; Maynez, J.; Rao, A.; Barnes, P.; Tay, Y.; Shazeer, N.; Prabhakaran, V.; Reif, E.; Du, N.; Hutchinson, B.; Pope, R.; Bradbury, J.; Austin, J.; Isard, M.; Gur-Ari, G.; Yin, P.; Duke, T.; Levskaya, A.; Ghemawat, S.; Dev, S.; Michalewski, H.; Garcia, X.; Misra, V.; Robinson, K.; Fedus, L.; Zhou, D.; Ippolito, D.; Luan, D.; Lim, H.; Zoph, B.; Spiridonov, A.; Sepassi, R.; Dohan, D.; Agrawal, S.; Omernick, M.; Dai, A. M.; Pillai, T. S.; Pellat, M.; Lewkowycz, A.; Moreira, E.; Child, R.; Polozov, O.; Lee, K.; Zhou, Z.; Wang, X.; Saeta, B.; Diaz, M.; Firat, O.; Catasta, M.; Wei, J.; Meier-Hellstern, K.; Eck, D.; Dean, J.; Petrov, S.; and Fiedel, N. 2022. PaLM: Scaling Language Modeling with Pathways. arXiv:2204.02311.

[20] Cook, D. J.; and Holder, L. B., eds. 2006. *Mining Graph Data*. John Wiley & Sons.

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

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

[23] Dohan, D.; Xu, W.; Lewkowycz, A.; Austin, J.; Bieber, D.; Lopes, R. G.; Wu, Y.; Michalewski, H.; Saurous, R. A.; Sohl-Dickstein, J.; Murphy, K.; and Sutton, C. 2022. Language Model Cascades. In *Beyond Bayes: Paths Towards Universal Reasoning Systems*, Workshop at ICML '22.

[24] Drori, I.; Zhang, S.; Shuttleworth, R.; Tang, L.; Lu, A.; Ke, E.; Liu, K.; Chen, L.; Tran, S.; Cheng, N.; Wang, R.; Singh, N.; Patti, T. L.; Lynch, J.; Shporer, A.; Verma, N.; Wu, E.; and Strang, G. 2022. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *Proceedings of the National Academy of Sciences*, 119(32): e2123433119.

[25] 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.

[26] 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.

[27] Friggeri, A.; Chelius, G.; and Fleury, E. 2011. Triangles to Capture Social Cohesion. In *Proceedings of the IEEE Third International Conference on Privacy, Security, Risk and Trust and IEEE Third International Conference on Social Computing*, PASSAT/SocialCom '11, 258–265.

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

[29] Fu, Y.; Peng, H.; Sabharwal, A.; Clark, P.; and Khot, T. 2022. Complexity-Based Prompting for Multi-Step Reasoning. arXiv:2210.00720.

[30] Gianinazzi, L.; Fries, M.; Dryden, N.; Ben-Nun, T.; Besta, M.; and Hoefler, T. 2021. Learning Combinatorial Node Labeling Algorithms. arXiv:2106.03594.

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

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

[33] 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.

[34] Hartmann, M.; and Sonntag, D. 2022. A survey on improving NLP models with human explanations. In *Proceedings of the First Workshop on Learning with Natural Language Supervision*, 40–47. Association for Computational Linguistics.

[35] Horváth, T.; Gärtner, T.; and Wrobel, S. 2004. Cyclic Pattern Kernels for Predictive Graph Mining. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, 158–167.

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

[37] Huang, W.; Xia, F.; Xiao, T.; Chan, H.; Liang, J.; Florence, P.; Zeng, A.; Tompson, J.; Mordatch, I.; Chebotar, Y.; Sermanet, P.; Brown, N.; Jackson, T.; Luu, L.; Levine, S.; Hausman, K.; and Ichter, B. 2022. Inner Monologue: Embodied Reasoning through Planning with Language Models. arXiv:2207.05608.

[38] Jiang, C.; Coenen, F.; and Zito, M. 2013. A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review*, 28(1): 75–105.

[39] Kim, G.; Baldi, P.; and McAleer, S. 2023. Language Models can Solve Computer Tasks. arXiv:2303.17491.

[40] Lertvittayakumjorn, P.; and Toni, F. 2021. Explanation-Based Human Debugging of NLP Models: A Survey. *Transactions of the Association for Computational Linguistics*, 9: 1508–1528.

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

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

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

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

[45] Madaan, A.; Tandon, N.; Gupta, P.; Hallinan, S.; Gao, L.; Wiegreffe, S.; Alon, U.; Dziri, N.; Prabhumoye, S.; Yang, Y.; Gupta, S.; Majumder, B. P.; Hermann, K.; Welleck, S.; Yazdanbakhsh, A.; and Clark, P. 2023. Self-Refine: Iterative Refinement with Self-Feedback. arXiv:2303.17651.

[46] Malewicz, G.; Austern, M. H.; Bik, A. J.; Dehnert, J. C.; Horn, I.; Leiser, N.; and Czajkowski, G. 2010. Pregel: A System for Large-Scale Graph Processing. In *Proceedings of the International Conference on Management of Data*, SIGMOD '10, 135–146. ACM.

[47] Ning, X.; Lin, Z.; Zhou, Z.; Wang, Z.; Yang, H.; and Wang, Y. 2023. Skeleton-of-Thought: Large Language Models Can Do Parallel Decoding. arXiv:2307.15337.

[48] Nye, M.; Andreassen, A. J.; Gur-Ari, G.; Michalewski, H.; Austin, J.; Bieber, D.; Dohan, D.; Lewkowycz, A.; Bosma, M.; Luan, D.; Sutton, C.; and Odena, A. 2021. Show Your Work: Scratchpads for Intermediate Computation with Language Models. arXiv:2112.00114.

[49] Paul, D.; Ismayilzada, M.; Peyrard, M.; Borges, B.; Bosselut, A.; West, R.; and Faltings, B. 2023. REFINER: Reasoning Feedback on Intermediate Representations. arXiv:2304.01904.

[50] Prat-Pérez, A.; Dominguez-Sal, D.; Brunat, J. M.; and Larriba-Pey, J.-L. 2012. Shaping Communities out of Triangles. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, 1677–1681.

[51] Qiao, S.; Ou, Y.; Zhang, N.; Chen, X.; Yao, Y.; Deng, S.; Tan, C.; Huang, F.; and Chen, H. 2023. Reasoning with Language Model Prompting: A Survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, ACL '23, 5368–5393. Association for Computational Linguistics.

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

[53] Radford, A.; Narasimhan, K.; Salimans, T.; and Sutskever, I. 2018. Improving Language Understanding by Generative Pre-Training. https://openai.com/research/language-unsupervised. Accessed: 2023-09-06.

[54] Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language Models are Unsupervised Multitask Learners. https://openai.com/research/better-language-models. Accessed: 2023-09-06.

[55] Robinson, I.; Webber, J.; and Eifrem, E. 2015. *Graph Databases: New Opportunities for Connected Data*. O'Reilly Media, 2nd edition.

[56] Sakr, S.; Bonifati, A.; Voigt, H.; Iosup, A.; Ammar, K.; Angles, R.; Aref, W.; Arenas, M.; Besta, M.; Boncz, P. A.; Daudjee, K.; Valle, E. D.; Dumbrava, S.; Hartig, O.; Haslhofer, B.; Hegeman, T.; Hidders, J.; Hose, K.; Iamnitchi, A.; Kalavri, V.; Kapp, H.; Martens, W.; Özsu, M. T.; Peukert, E.; Plantikow, S.; Ragab, M.; Ripeanu, M. R.; Salihoglu, S.; Schulz, C.; Selmer, P.; Sequeda, J. F.; Shinavier, J.; Szárnyas, G.; Tommasini, R.; Tumeo, A.; Uta, A.; Varbanescu, A. L.; Wu, H.-Y.; Yakovets, N.; Yan, D.; and Yoneki, E. 2021. The Future is Big Graphs: A Community View on Graph Processing Systems. *Commun. ACM*, 64(9): 62–71.

[57] 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.

[58] Schaeffer, S. E. 2007. Graph clustering. *Computer Science Review*, 1(1): 27–64.

[59] 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.

[60] Shinn, N.; Labash, B.; and Gopinath, A. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. arXiv:2303.11366.

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

[62] Teixeira, C. H. C.; Fonseca, A. J.; Serafini, M.; Siganos, G.; Zaki, M. J.; and Aboulnaga, A. 2015. Arabesque: A System for Distributed Graph Mining. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, 425–440. ACM.

[63] Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; Rodriguez, A.; Joulin, A.; Grave, E.; and Lample, G. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971.

[64] Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; Bikel, D.; Blecher, L.; Ferrer, C. C.; Chen, M.; Cucurull, G.; Esiobu, D.; Fernandes, J.; Fu, J.; Fu, W.; Fuller, B.; Gao, C.; Goswami, V.; Goyal, N.; Hartshorn, A.; Hosseini, S.; Hou, R.; Inan, H.; Kardas, M.; Kerkez, V.; Khabsa, M.; Kloumann, I.; Korenev, A.; Koura, P. S.; Lachaux, M.-A.; Lavril, T.; Lee, J.; Liskovich, D.; Lu, Y.; Mao, Y.; Martinet, X.; Mihaylov, T.; Mishra, P.; Molybog, I.; Nie, Y.; Poulton, A.; Reizenstein, J.; Rungta, R.; Saladi, K.; Schelten, A.; Silva, R.; Smith, E. M.; Subramanian, R.; Tan, X. E.; Tang, B.; Taylor, R.; Williams, A.; Kuan, J. X.; Xu, P.; Yan, Z.; Zarov, I.; Zhang, Y.; Fan, A.; Kambadur, M.; Narang, S.; Rodriguez, A.; Stojnic, R.; Edunov, S.; and Scialom, T. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288.

[65] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems (NIPS '17)*, volume 30. Curran Associates.

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

[67] Wang, X.; Wei, J.; Schuurmans, D.; Le, Q. V.; Chi, E. H.; Narang, S.; Chowdhery, A.; and Zhou, D. 2023. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *Proceedings of the Eleventh International Conference on Learning Representations*, ICLR '23.

[68] Wang, Z.; Cai, S.; Chen, G.; Liu, A.; Ma, X.; and Liang, Y. 2023. 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.

[69] Wang, Z.; Zhang, G.; Yang, K.; Shi, N.; Zhou, W.; Hao, S.; Xiong, G.; Li, Y.; Sim, M. Y.; Chen, X.; Zhu, Q.; Yang, Z.; Nik, A.; Liu, Q.; Lin, C.; Wang, S.; Liu, R.; Chen, W.; Xu, K.; Liu, D.; Guo, Y.; and Fu, J. 2023. Interactive Natural Language Processing. arXiv:2305.13246.

[70] Wang, Z. J.; Choi, D.; Xu, S.; and Yang, D. 2021. Putting Humans in the Natural Language Processing Loop: A Survey. In *Proceedings of the First Workshop on Bridging Human-Computer Interaction and Natural Language Processing*, 47–52. Association for Computational Linguistics.

[71] Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Chi, E.; Le, Q.; and Zhou, D. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903.

[72] 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.

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

[74] 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.

[75] Xie, Y.; Kawaguchi, K.; Zhao, Y.; Zhao, X.; Kan, M.-Y.; He, J.; and Xie, Q. 2023. Self-Evaluation Guided Beam Search for Reasoning. In *Advances in Neural Information Processing Systems (NeurIPS '23)*, volume 36. Curran Associates.

[76] 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.

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

[78] Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K. R.; and Cao, Y. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *Proceedings of the Eleventh International Conference on Learning Representations*, ICLR '23.

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

[80] Zelikman, E.; Wu, Y.; Mu, J.; and Goodman, N. 2022. STaR: Bootstrapping Reasoning With Reasoning. In *Advances in Neural Information Processing Systems (NeurIPS '22)*, volume 35, 15476–15488. Curran Associates.

[81] Zhang, S.; Chen, Z.; Shen, Y.; Ding, M.; Tenenbaum, J. B.; and Gan, C. 2023. Planning with Large Language Models for Code Generation. In *Proceedings of the Eleventh International Conference on Learning Representations*, ICLR '23.

[82] 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.

[83] Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; and Sun, M. 2020. Graph neural networks: A review of methods and applications. *AI Open*, 1: 57–81.

[84] Zhou, Y.; Muresanu, A. I.; Han, Z.; Paster, K.; Pitis, S.; Chan, H.; and Ba, J. 2022. Large Language Models Are Human-Level Prompt Engineers. arXiv:2211.01910.

[85] 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 *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, ACL '23, 4471–4485. Association for Computational Linguistics.

A Positive Score Evaluation

The following figures plot the same data as Figures 5 and 6 respectively, however use the ”positive score” described in Sections 5.1 and 5.2.

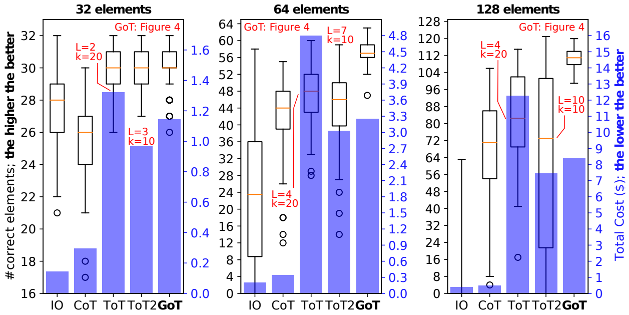


Figure 9: Accuracy and cost in sorting tasks with ChatGPT-3.5. L and k indicate the structure of ToT (see Sections 3.2 and 6).

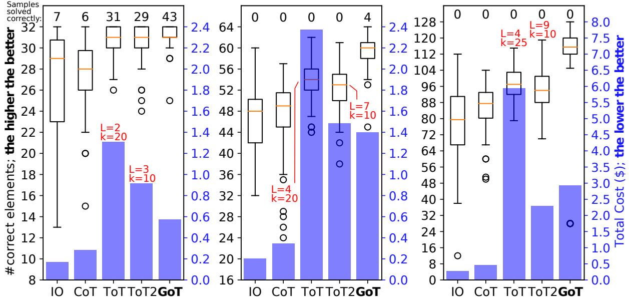


Figure 10: Accuracy and cost in set intersection with ChatGPT-3.5. L and k indicate the structure of ToT (see Sections 3.2 and 6).

B Example Prompts - Sorting

We present the prompts only for the sorting of 32-element lists, as those for 64-element and 128-element lists are identical, except for the *split_prompt* where the number of elements in the one-shot example matches the problem size.

For sorting, we employ three distinct types of operations that interact with the LLM, each with its corresponding prompts. First, there is the **Generate** operation, utilizing the *sort_prompt* to guide the LLM in sorting a provided list of values, and the *split_prompt* to direct the LLM to split a specified list into a designated number of sublists. Next, the **Improve** operation employs the *improve_prompt* to instruct the LLM to refine a sorted list if it detects mistakes. Finally, the **Aggregate** operation leverages the *merge_prompt* to guide the LLM in merging two pre-sorted lists into a single sorted list.

First, we present the prompt stubs (Table 3), serving as templates to dynamically generate appropriate prompts at runtime. For clarity, we display their corresponding few-shot examples separately in Table 4. Following this, we outline the LLM interactions throughout the process of solving the sorting use case (Table 5 - Table 9).

Table 3: Prompt stubs for the sorting tasks; parameters in single curly brackets will be substituted at runtime.

sort_prompt: <Instruction> Sort the following list of numbers in ascending order. Output only the sorted list of numbers, no additional text. </Instruction>

<Examples> See Table 4 </Examples>

Input: {input_list}

split_prompt (32 elements): <Instruction> Split the following list of 32 numbers into 2 lists of 16 numbers each, the first list should contain the first 16 numbers and the second list the second 16 numbers.

Only output the final 2 lists in the following format without any additional text or thoughts!:

```
{{
  "List 1": [3, 4, 3, 5, 7, 8, 1, ...],
  "List 2": [2, 9, 2, 4, 7, 1, 5, ...]
}}
```

</Instruction>

<Examples> See Table 4 </Examples>

Input: {input_list}

improve_prompt: <Instruction> The following two lists represent an unsorted list of numbers and a sorted variant of that list. The sorted variant is not correct. Fix the sorted variant so that it is correct. Make sure that the output list is sorted in ascending order, has the same number of elements as the input list ({length}), and contains the same elements as the input list.</Instruction>

<Approach>

To fix the incorrectly sorted list follow these steps:

1. For each number from 0 to 9, compare the frequency of that number in the incorrectly sorted list to the frequency of that number in the input list.
2. Iterate through the incorrectly sorted list and add or remove numbers as needed to make the frequency of each number in the incorrectly sorted list match the frequency of that number in the input list.

</Approach>

<Examples> See Table 4 </Examples>

Input: {input_list}

Incorrectly Sorted: {sorted_list}

merge_prompt: <Instruction> Merge the following 2 sorted lists of length {length} each, into one sorted list of length {length_combined} using a merge sort style approach. Only output the final merged list without any additional text or thoughts!: </Instruction>

<Approach>

To merge the two lists in a merge-sort style approach, follow these steps:

1. Compare the first element of both lists.
2. Append the smaller element to the merged list and move to the next element in the list from which the smaller element came.
3. Repeat steps 1 and 2 until one of the lists is empty.
4. Append the remaining elements of the non-empty list to the merged list.

</Approach>

Merge the following two lists into one sorted list:

1. {input_list1}
2. {input_list2}

Merged list:

Table 4: Few-shot examples for each prompt used for the sorting tasks; some lists are truncated for brevity.

sort_prompt:

<Examples>

Input: [5, 1, 0, 1, 2, 0, 4, 8, 1, 9, 5, 1, 3, 3, 9, 7]

Output: [0, 0, 1, 1, 1, 1, 2, 3, 3, 4, 5, 5, 7, 8, 9, 9]

Input: [3, 7, 0, 2, 8, 1, 2, 2, 2, 4, 7, 8, 5, 5, 3, 9, 4, 3, ... (Omitted 14/32 numbers)]

Output: [0, 0, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 4, 4, 4, ... (Omitted 14/32 numbers)]

Input: [4, 4, 9, 7, 9, 7, 0, 0, 4, 9, 1, 7, 9, 5, 8, 7, 5, 6, ... (Omitted 46/64 numbers)]

Output: [0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, ... (Omitted 46/64 numbers)]

</Examples>

split_prompt (32 elements):

<Examples>

Input: [9, 6, 7, 7, 2, 0, 2, 2, 3, 5, 0, 9, 2, 2, 4, 4, 5, 2, ... (Omitted 14/32 numbers)]

Output:

{{
  ”List 1”: [9, 6, 7, 7, 2, 0, 2, 2, 3, 5, 0, 9, 2, 2, 4, 4],
  ”List 2”: [5, 2, 5, 1, 2, 8, 3, 8, 3, 9, 6, 0, 4, 2, 2, 3]
}}

</Examples>

improve_prompt:

<Examples>

Input: [3, 7, 0, 2, 8, 1, 2, 2, 2, 4, 7, 8, 5, 5, 3, 9]

Incorrectly Sorted: [0, 0, 0, 0, 0, 1, 2, 2, 3, 3, 4, 4, 4, 5, 5, 7, 7, 8, 8, 9, 9, 9, 9]

Reason: The incorrectly sorted list contains four extra 0s, two extra 4s and three extra 9s and is missing two 2s.

Output: [0, 1, 2, 2, 2, 2, 3, 3, 4, 5, 5, 7, 7, 8, 8, 9]

Input: [6, 4, 5, 7, 5, 6, 9, 7, 6, 9, 4, 6, 9, 8, 1, 9, 2, 4, ... (Omitted 14/32 numbers)]

Incorrectly Sorted: [0, 1, 1, 2, 2, 3, 4, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, ... (Omitted 14/32 numbers)]

Reason: The incorrectly sorted list contains two extra 4s and is missing two 6s and one 9.

Output: [0, 1, 1, 2, 2, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 6, ... (Omitted 14/32 numbers)]

Input: [4, 4, 9, 7, 9, 7, 0, 0, 4, 9, 1, 7, 9, 5, 8, 7, 5, 6, ... (Omitted 46/64 numbers)]

Incorrectly Sorted: [0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, ... (Omitted 46/64 numbers)]

Reason: The incorrectly sorted list contains one extra 8 and is missing two 2s, one 3, three 4s, two 5s, one 6, six 7s and one 9.

Output: [0, 0, 0, 0, 0, 0, 0, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, ... (Omitted 46/64 numbers)]

</Examples>

merge_prompt: -

Table 5: Sorting of a 32 element list: Execution plan (**GoO**)

GoO:

1. Split the input list into two sub-lists of equal size (split_prompt)
2. For each sub-list: Sort the sub-list (sort_prompt) five times; score each sort attempt; keep the best
3. Merge the sorted sub-lists into one fully sorted list (merge_prompt) 10 times; score each merge attempt; keep the best
4. Fix any potential mistakes in the sorted list (improve_prompt) 10 times; score each improvement attempt; keep the best

---

Table 6: Sorting of a 32 element list: **Step 1 (Prompt/Response)**

Step 1 – Prompt:

<Instruction> Split the following list of 32 numbers into 2 lists of 16 numbers each, the first list should contain the first 16 numbers and the second list the second 16 numbers.

Only output the final 2 lists in the following format without any additional text or thoughts!:

```
{{
  "List 1": [3, 4, 3, 5, 7, 8, 1, ...],
  "List 2": [2, 9, 2, 4, 7, 1, 5, ...]
}}
```

</Instruction>

<Examples> See Table 4 </Examples>

Input: `[8, 7, 1, 1, 1, 1, 3, 3, 0, 9, 4, 1, 0, 2, 5, 1, 0, 5, 6, 7, 1, 4, 5, 9, 4, 6, 2, 5, 8, 6, 2, 6]`

Step 1 – Response (Fully Correct):

```
{{
  "List 1": [8, 7, 1, 1, 1, 1, 3, 3, 0, 9, 4, 1, 0, 2, 5, 1],
  "List 1": [0, 5, 6, 7, 1, 4, 5, 9, 4, 6, 2, 5, 8, 6, 2, 6]
}}
```

Table 7: Sorting of a 32 element list: **Step 2 (Prompts/Responses)**

Step 2a – Prompt:

<Instruction> Sort the following list of numbers in ascending order. Output only the sorted list of numbers, no additional text. </Instruction>

<Examples> See Table 4 </Examples>

Input: `[8, 7, 1, 1, 1, 1, 3, 3, 0, 9, 4, 1, 0, 2, 5, 1]`

Step 2a – 5 Responses:

1. **`[0, 0, 1, 1, 1, 1, 1, 1, 2, 3, 3, 4, 5, 7, 8, 9]`** **(Fully Correct)**
2. `[0, 0, 1, 1, 1, 1, 1, 2, 3, 3, 4, 5, 7, 8, 9]` (1 Error - Missing one 1)
3. `[0, 0, 1, 1, 1, 1, 1, 2, 3, 3, 4, 5, 7, 8, 9]` (1 Error - Missing one 1)
4. `[0, 0, 1, 1, 1, 1, 1, 2, 3, 3, 4, 5, 7, 8, 9]` (1 Error - Missing one 1)
5. `[0, 0, 1, 1, 1, 1, 1, 2, 3, 3, 4, 5, 7, 8, 9]` (1 Error - Missing one 1)

Step 2b – Prompt:

<Instruction> Sort the following list of numbers in ascending order. Output only the sorted list of numbers, no additional text. </Instruction>

<Examples> See Table 4 </Examples>

Input: `[0, 5, 6, 7, 1, 4, 5, 9, 4, 6, 2, 5, 8, 6, 2, 6]`

Step 2b – 5 Responses:

1. `[0, 1, 2, 2, 4, 4, 5, 5, 5, 6, 6, 6, 7, 8, 9]` (1 Error - Missing one 6)
2. `[0, 1, 2, 2, 4, 4, 5, 5, 5, 6, 6, 6, 7, 8, 9]` (1 Error - Missing one 6)
3. `[0, 1, 2, 2, 4, 4, 5, 5, 5, 6, 6, 6, 7, 8, 9]` (1 Error - Missing one 6)
4. **`[0, 1, 2, 2, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 8, 9]`** **(Fully Correct)**
5. `[0, 1, 2, 2, 4, 4, 5, 5, 5, 6, 6, 6, 7, 8, 9]` (1 Error - Missing one 6)

Table 8: Sorting of a 32 element list: **Step 3 (Prompt/Responses)**

Step 3 – Prompt:

<Instruction> Merge the following 2 sorted lists of length 16 each, into one sorted list of length 32 using a merge sort style approach. Only output the final merged list without any additional text or thoughts!: </Instruction>

<Approach>

To merge the two lists in a merge-sort style approach, follow these steps:

1. Compare the first element of both lists.
2. Append the smaller element to the merged list and move to the next element in the list from which the smaller element came.
3. Repeat steps 1 and 2 until one of the lists is empty.
4. Append the remaining elements of the non-empty list to the merged list.

</Approach>

Merge the following two lists into one sorted list:

1. `[0, 0, 1, 1, 1, 1, 1, 1, 2, 3, 3, 4, 5, 7, 8, 9]`
2. `[0, 1, 2, 2, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 8, 9]`

Merged list:

Step 3 – 10 Responses:

1. `[0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9]`
   (2 Errors - Missing one 1 and one 5)
2. `[0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9]`
   (2 Errors - Missing two 1s)
3. `[0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9]`
   (2 Errors - Missing one 1 and got one extra 6)
4. `[0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 6, 7, 7, 8, 8, 9]`
   (3 Errors - Missing one 1, one 9 and got one extra 6)
5. `[0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9]`
   (2 Errors - Missing one 1 and got one extra 6)
6. `[0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9]`
   (2 Errors - Missing one 1 and one 5)
7. `[0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9]`
   (2 Errors - Missing one 1 and got one extra 6)
8. **`[0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9]`**
   (1 Error - Missing one 1)
9. `[0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9]`
   (1 Error - Missing one 1)
10. `[0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9]`
   (2 Errors - Missing one 1 and one 5)

Table 9: Sorting of a 32 element list: **Step 4 (Prompt/Responses)**

Step 4 – Prompt:

<Instruction> The following two lists represent an unsorted list of numbers and a sorted variant of that list. The sorted variant is not correct. Fix the sorted variant so that it is correct. Make sure that the output list is sorted in ascending order, has the same number of elements as the input list (32), and contains the same elements as the input list.</Instruction>

<Approach>

To fix the incorrectly sorted list follow these steps:

1. For each number from 0 to 9, compare the frequency of that number in the incorrectly sorted list to the frequency of that number in the input list.
2. Iterate through the incorrectly sorted list and add or remove numbers as needed to make the frequency of each number in the incorrectly sorted list match the frequency of that number in the input list.

</Approach>

<Examples> See Table 4 </Examples>

Input: `[8, 7, 1, 1, 1, 1, 3, 3, 0, 9, 4, 1, 0, 2, 5, 1, 0, 5, 6, 7, 1, 4, 5, 9, 4, 6, 2, 5, 8, 6, 2, 6]`

Incorrectly Sorted: `[0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9]`

Step 4 – 10 Responses:

1. Reason: The incorrectly sorted list is missing two 0s, one 3, one 4, one 6 and one 9. The incorrectly sorted list also contains one extra 1, one extra 2, and one extra 5.
   `Output: [0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9]` 4 Errors - Missing one 1, two 4s and one 5
   Note: The output list has the same number of elements as the input list, is sorted in ascending order, and contains the same elements as the input list.
2. `Output: [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9]` 1 Error - One extra 6
3. `Output: [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 7, 7, 7, 8, 8, 8, 9, 9, 9]` 6 Errors - One extra 2, 5, 6, 7, 8 and 9
4. Reason: The incorrectly sorted list is missing two 9s and one 0.
   `Output: [0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9, 9, 9]` 3 Errors - Missing one 1 and has two extra 9s
5. Reason: The incorrectly sorted list is missing one 0, one 3, one 4, four 6s, and one 9.
   `Output: [0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 8, 8, 8, 9, 9, 9, 9]` 10 Errors - Missing one 1, 2, has four extra 6s, one extra 7, 8 and two extra 9s
6. `Output: [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 9, 9, 9, 9]` 8 Errors - Has one extra 2, 3, 6, two extra 7s, one extra 8 and two extra 9s
7. `Output: [0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9]` 2 Errors - Missing one 1 and has one extra 4
8. Reason: The incorrectly sorted list is missing three 0s, two 3s, two 4s, one 5, two 6s, and one 9.
   `Output: [0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 7, 7, 8, 8, 9]` 5 Errors - Missing one 1, 9, has one extra 3 and two extra 6s
9. Reason: The incorrectly sorted list is missing one 0, three 2s, two 3s, two 4s, three 5s, two 6s, one 7, and one 9.
   `Output: [0, 0, 0, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 7, 8, 8, 9, 9]` 8 Errors - Missing one 1, one 7, has two extra 2s, two extra 3s, one extra 5 and 6
10. **Reason: The incorrectly sorted list is missing three 1s, one 3, one 4, two 5s, one 6, one 7, one 8, and one 9.**
    `Output: [0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9]` (Fully Correct)

Final Result (Correctly Sorted):

`[0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 7, 7, 8, 8, 9, 9]`

C Example Prompts - Set Intersection

We present the prompts only for the intersection of two 32-element sets, as those for 64-element and 128-element sets are identical, except for the *split_prompt* where the size of the split is adjusted proportionally.

For set intersection, we employ two distinct types of operations that interact with the LLM, each with its corresponding prompts. First, there is the **Generate** operation, utilizing the *intersect_prompt* to guide the LLM in intersecting two input sets, and the *split_prompt* to direct the LLM to split a specified set into a designated number of distinct subsets. Second, the **Aggregate** operation leverages the *merge_prompt* to guide the LLM in combining two sets into one.

First, we present the prompt stubs (Table 10), serving as templates to dynamically generate appropriate prompts at runtime. For clarity, we display their corresponding few-shot examples separately in Table 11. Following this, we outline the LLM interactions throughout a complete set intersection process (Table 12 - Table 15).

Table 10: Prompt stubs for the set intersection tasks; parameters in single curly brackets will be substituted at runtime.

---

intersect_prompt: <Instruction> Find the intersection of two sets of numbers. Output only the set of numbers that are present in both sets, no additional text.</Instruction>

<Examples> See Table 11 </Examples>

Input Set 1: {set1}

Input Set 2: {set2}

---

split_prompt (32 elements): <Instruction> Split the following list of 32 numbers into 2 lists of 16 numbers each, the first list should contain the first 16 numbers and the second list the second 16 numbers.

Only output the 2 lists in the following format without any additional text or thoughts!

```
{{
  "List 1": [13, 16, 30, 6, 21, 7, 31, ...],
  "List 2": [25, 24, 10, 4, 27, 0, 14, ...]
}}
```

</Instruction>

<Examples> See Table 11 </Examples>

Input: {input}

---

merge_prompt: <Instruction> Merge the following 2 lists into one list by appending the second list to the first list.

Only output the final list without any additional text or thoughts! </Instruction>

List 1: {input1}

List 2: {input2}

Table 11: Few-shot examples for each prompt used for the set intersection tasks; some lists are truncated for brevity.

---

intersect_prompt:

<Examples>

`Input Set 1: [13, 16, 30, 6, 21, 7, 31, 15, 11, 1, 24, 10, 9, 3, 20, 8]`

`Input Set 2: [25, 24, 10, 4, 27, 0, 14, 12, 8, 2, 29, 20, 17, 19, 26, 23]`

`Output: [24, 10, 20, 8]`

`Input Set 1: [26, 40, 42, 57, 15, 31, 5, 32, 11, 4, 24, 28, 51, 54,` ... (Omitted 18/32 numbers)`]`

`Input Set 2: [16, 60, 36, 48, 0, 15, 5, 19, 46, 24, 1, 6, 61, 10,` ... (Omitted 18/32 numbers)`]`

`Output: [40, 15, 5, 24, 35, 59, 16, 63]`

`Input Set 1: [115, 61, 35, 103, 90, 117, 86, 44, 63, 45, 40, 30, 74, 33,` ... (Omitted 50/64 numbers)`]`

`Input Set 2: [13, 35, 20, 96, 34, 18, 47, 127, 126, 9, 21, 16, 77, 22,` ... (Omitted 50/64 numbers)`]`

`Output: [115, 35, 90, 117, 63, 40, 30, 33, 15, 89, 50, 12, 2, 84, 7, 57, 96]`

</Examples>

---

split_prompt (32 elements):

<Examples>

`Input: [26, 40, 42, 57, 15, 31, 5, 32, 11, 4, 24, 28, 51, 54,` ... (Omitted 18/32 numbers)`]`

`Output:`

```
{{
  "List 1": [26, 40, 42, 57, 15, 31, 5, 32, 11, 4, 24, 28, 51, 54, 12, 22],
  "List 2": [33, 35, 7, 13, 2, 59, 8, 23, 43, 16, 29, 55, 25, 63, 21, 18]
}}
```

</Examples>

---

merge_prompt: -

Table 12: Intersection of two 32-element sets: Execution plan (**GoO**)

GoO:

1. Split the second input set into two sub-sets of equal size (split_prompt)
2. For each sub-set: Intersect the sub-set with the first input set (intersect_prompt) five times; score each sort attempt; keep the best
3. Merge the resulting intersections into one full intersection set (merge_prompt) 10 times; score each merge attempt; keep the best

---

Table 13: Intersection of two 32-element sets: **Step 1 (Prompt/Response)**

Step 1 – Prompt:

<Instruction> Split the following list of 32 numbers into 2 lists of 16 numbers each, the first list should contain the first 16 numbers and the second list the second 16 numbers.

Only output the 2 lists in the following format without any additional text or thoughts!

```
{{
  "List 1": [13, 16, 30, 6, 21, 7, 31, ...],
  "List 2": [25, 24, 10, 4, 27, 0, 14, ...]
}}
```

</Instruction>

<Examples> See Table 11 </Examples>

Input: `[31, 11, 4, 63, 38, 58, 59, 24, 61, 14, 32, 39, 27, 46, 48, 19, 52, 57, 50, 56, 3, 2, 53, 29, 5, 37, 62, 41, 36, 12, 49, 16]`

Step 1 – Response (Fully Correct):

```
{{
  "List 1": [31, 11, 4, 63, 38, 58, 59, 24, 61, 14, 32, 39, 27, 46, 48, 19],
  "List 1": [52, 57, 50, 56, 3, 2, 53, 29, 5, 37, 62, 41, 36, 12, 49, 16]
}}
```

Table 14: Intersection of two 32-element sets: **Step 2 (Prompts/Responses)**

Step 2a – Prompt:

<Instruction> Find the intersection of two sets of numbers. Output only the set of numbers that are present in both sets, no additional text. </Instruction>

<Examples> See Table 11 </Examples>

Input Set 1: `[11, 60, 1, 49, 21, 33, 14, 56, 54, 15, 23, 40, 45, 22, 7, 28, 20, 46, 51, 6, 34, 37, 3, 50, 17, 8, 25, 0, 35, 47, 18, 19]`

Input Set 2: `[31, 11, 4, 63, 38, 58, 59, 24, 61, 14, 32, 39, 27, 46, 48, 19]`

Step 2a – 5 Responses:

1. `[11, 14, 46, 14, 19]` (1 Error - Duplicated 14)
2. **`Output: [11, 14, 46, 19]` (Fully Correct)**
3. `[11, 14, 46, 14, 19]` (1 Error - Duplicated 14)
4. `[11, 14, 46, 19]` (Fully Correct)
5. `[11, 14, 46, 14, 19]` (1 Error - Duplicated 14)

Step 2b – Prompt:

<Instruction> Find the intersection of two sets of numbers. Output only the set of numbers that are present in both sets, no additional text. </Instruction>

<Examples> See Table 11 </Examples>

Input Set 1: `[11, 60, 1, 49, 21, 33, 14, 56, 54, 15, 23, 40, 45, 22, 7, 28, 20, 46, 51, 6, 34, 37, 3, 50, 17, 8, 25, 0, 35, 47, 18, 19]`

Input Set 2: `[52, 57, 50, 56, 3, 2, 53, 29, 5, 37, 62, 41, 36, 12, 49, 16]`

Step 2b – 5 Responses:

1. `[50, 56, 49]` (2 Errors - Missing 3 and 37)
2. `[50, 56, 37, 49]` (1 Error - Missing 3)
3. **`[56, 49, 37, 3, 50]` (Fully Correct)**
4. `[50, 56, 37, 49]` (1 Error - Missing 3)
5. `[50, 56, 37, 49]` (1 Error - Missing 3)

Table 15: Intersection of two 32-element sets: **Step 3 (Prompt/Responses)**

Step 3 – Prompt:

<Instruction> Merge the following 2 lists into one list by appending the second list to the first list.

Only output the final list without any additional text or thoughts! </Instruction>

List 1: `[11, 14, 46, 19]`

List 2: `[56, 49, 37, 3, 50]`

Step 3 – 10 Responses:

1. **`[11, 14, 46, 19, 56, 49, 37, 3, 50]` (Fully Correct)**
2. `[11, 14, 46, 19, 56, 49, 37, 3, 50]` (Fully Correct)
3. `[11, 14, 46, 19, 56, 49, 37, 3, 50]` (Fully Correct)
4. `[11, 14, 46, 19, 56, 49, 37, 3, 50]` (Fully Correct)
5. `[11, 14, 46, 19, 56, 49, 37, 3, 50]` (Fully Correct)
6. `[11, 14, 46, 19, 56, 49, 37, 3, 50]` (Fully Correct)
7. `[11, 14, 46, 19, 56, 49, 37, 3, 50]` (Fully Correct)
8. `[11, 14, 46, 19, 56, 49, 37, 3, 50]` (Fully Correct)
9. `[11, 14, 46, 19, 56, 49, 37, 3, 50]` (Fully Correct)
10. `[11, 14, 46, 19, 56, 49, 37, 3, 50]` (Fully Correct)

Final Result (Correct Intersection):

`[11, 14, 46, 19, 56, 49, 37, 3, 50]`

D Example Prompts - Keyword Counting

We present the prompts only for `GoT4` of the keyword counting task, as those used for `GoT8` and `GoTx` are identical, except for minor differences in the *split_prompt* where the size of the split is adjusted.

For keyword counting, we employ three distinct types of operations that interact with the LLM, each with its corresponding prompts. First, there is the **Generate** operation, utilizing the *count_prompt* to guide the LLM in counting the keywords in a text, and the *split_prompt* to direct the LLM to split a given text into a number of passages. Next, the **Aggregate** operation leverages the *merge_prompt* to guide the LLM in merging two dictionaries of counted keywords into one. Finally, the **ValidateAndImprove** operation employs the *improve_merge_prompt* to instruct the LLM to correct mistakes that were made in a previous Aggregate operation.

We present the prompt stubs (Table 16 - Table 17), serving as templates to dynamically generate appropriate prompts at runtime. For clarity, we display their corresponding few-shot examples separately in Table 18 and Table 19. Following this, we outline the LLM interactions throughout a complete keyword counting process (Table 20 - Table 28).

Table 16: Prompt stubs for the keyword counting task; parameters in single curly brackets will be substituted at runtime.

---

count_prompt: <Instruction> Count the frequency of how many times each country is explicitly named in the input text. You can generate any intermediate lists and states, but the final output should only contain the frequency of each country that appears at least once in the following json format, prefixed with ”Output: ” (make sure to keep the same spelling for each country in the output as in the input text):

{{

```
  "country1": frequency1,
  "country2": frequency2,
  ...
```

}}

</Instruction>

<Approach>

To count the frequency for each country follow these steps:

1. Split the input passage into four paragraphs of similar length.
2. Count the frequency of each country in each paragraph.
3. Combine the frequencies of each country from each paragraph by adding them together.

</Approach>

<Examples> See Table 18 </Examples>

Input: {input_text}

---

split_prompt: <Instruction> Split the following input text into 4 paragraphs of approximately same length.

Only output the final 4 paragraphs in the following format without any additional text or thoughts:

{{

```
  "Paragraph 1": "Some paragraph text ...",
  "Paragraph 2": "Some paragraph text ...",
  "Paragraph 3": "Some paragraph text ...",
  "Paragraph 4": "Some paragraph text ..."
```

}}

</Instruction>

<Example> See Table 19 </Example>

Input: {input_text}

Table 17: Prompt stubs for the keyword counting task *continued*; parameters in single curly brackets will be substituted at runtime.

---

merge_prompt: <Instruction> Combine the following 2 dictionaries, each containing the frequency of countries in a text, into a single dictionary. Simply add the frequencies together for each country and if a country is not present in one of the dictionaries, add it to the final dictionary with the frequency from the other dictionary.

Only output the final merged dictionary without any additional text or thoughts! </Instruction>

<Approach>

To combine the 2 dictionaries into single one, follow these steps:

1. Create a new dictionary to store the combined frequencies.
2. Iterate through the keys of the first dictionary and add the frequency of each country to the new dictionary.
3. Iterate through the keys of the second dictionary and add the frequency of each country to the new dictionary and if it is already present, add the frequency to the existing value.

</Approach>

Combine the following 2 dictionaries into a single dictionary:

{dictionary_1}

{dictionary_2}

Combined Output:

---

improve_merge_prompt: <Instruction> The following 2 dictionaries were combined into the third dictionary below. However, some mistakes occured and the third dictionary is incorrect. Please fix the third dictionary so that it contains the correct frequencies for each country. The correct frequencies are the sum of the frequencies from the first 2 dictionaries. If a country is not present in one of the dictionaries, add it to the final dictionary with the frequency from the other dictionary.

</Instruction>

<Example> See Table 19 </Example>
Dictionary 1: {dictionary_1}

Dictionary 2: {dictionary_2}

Incorrectly Combined Dictionary: {dictionary_incorrect}

Output:

Table 18: Few-shot examples for **count** prompt used for the keyword counting task; some paragraphs and dictionaries are truncated and formatting is slightly adjusted for brevity.

---

count_prompt:
<Examples>
`Input:` Alexandra boarded the first flight of her grand journey, starting from Canada. With a globe-trotting ... (Omitted)
`Paragraphs:`
Alexandra boarded the first flight of her grand journey, starting from Canada. With a globe-trotting itinerary ... (Omitted)
Her first stop was Mexico, where she marveled at the Mayan ruins. From there, she explored the rainforests ... (Omitted)
`Sublist frequencies:`
`{{ "Canada": 1 }}`
`{{ "Mexico": 1, "Brazil": 1, "Argentina": 1 }}`
`Output: {{ "Canada": 1, "Mexico": 1, "Brazil": 1, "Argentina": 1 }}`

`Input:` The adventure led him to the peaks of Peru where he trekked to see the mysteries of Machu Picchu ... (Omitted)
`Paragraphs:`
The adventure led him to the peaks of Peru where he trekked to see the mysteries of Machu Picchu. He then ... (Omitted)
A quick detour to Uruguay and Paraguay allowed him to experience the vibrancy of the local cultures before ... (Omitted)
`Sublists:`
`{{ "Peru": 1, "Chile": 1 }}`
`{{ "Uruguay": 1, "Paraguay": 1, "Canada": 1, "Peru": 1, "Brazil": 1, "Mexico": 1 }}`
`Output: {{ "Peru": 2, "Chile": 1, "Uruguay": 1, "Paraguay": 1, "Canada": 1, "Brazil": 1, "Mexico": 1 }}`

`Input:` Journeying westward, she admired the art in Italy and sipped coffee in France. The music of ... (Omitted)
`Paragraphs:`
Journeying westward, she admired the art in Italy and sipped coffee in France.
The music of Spain and the history of Greece deepened her love for Europe. The Nordic beauty of Norway, ... (Omitted)
She danced in Ireland, explored castles in Scotland, and marveled at the architecture in Germany and Russia.
Italy, Norway, Sweden and Germany will always stay her favourite destinations to visit.
`Sublists:`
`{{ "Italy": 1, "France": 1 }}`
`{{ "Spain": 1, "Greece": 1, "Norway": 1, "Sweden": 1, "Finland": 1, "Denmark": 1 }}`
`{{ "Ireland": 1, "Scotland": 1, "Germany": 1, "Russia": 1 }}`
`{{ "Italy": 1, "Norway": 1, "Sweden": 1, "Germany": 1 }}`
`Output: {{ "Italy": 2, "France": 1, "Spain": 1, "Greece": 1, "Norway": 2, "Sweden": 2,` ... (Omitted) `}}`
</Examples>

Table 19: Few-shot examples for **split**, **merge** and **improve_merge** prompts used for the keyword counting task; some paragraphs and dictionaries are truncated and formatting is slightly adjusted for brevity.

---

split_prompt:

<Examples>

`Input:` Journeying westward, she admired the art in Italy and sipped coffee in France. The music of Spain and the history of Greece deepened her love for Europe. The Nordic beauty of Norway, Sweden, Finland, and Denmark took her breath away. She danced in Ireland, explored castles in Scotland, and marveled at the architecture in Germany and Russia. Italy, Norway, Sweden and Germany will always stay her favourite destinations to visit.

`Output:`

{{

`"Paragraph 1": "Journeying westward, she admired the art in Italy and sipped coffee in France. ",`

`"Paragraph 2": "The music of Spain and the history of Greece deepened her love for` ... (Omitted)",

`"Paragraph 3": "She danced in Ireland, explored castles in Scotland, and marveled` ... (Omitted)",

`"Paragraph 4": "Italy, Norway, Sweden and Germany will always stay her favourite` ... (Omitted)"

}}

</Examples>

---

merge_prompt: -

---

improve_merge_prompt:

<Example>

Dictionary 1: {{ `"Peru": 2, "Chile": 1, "Uruguay": 1, "Paraguay": 1` }}

Dictionary 2: {{ `"Peru": 1, "Argentina": 1, "Canada": 1, "Chile": 3, "Germany": 2` }}

Incorrectly Combined Dictionary:

{{ `"Peru": 3, "Chile": 2, "Uruguay": 1, "Paraguay": 1, "Argentina": 1, "Chile": 3, "Germany": 2` }}

Output:

{{ `"Peru": 3, "Chile": 4, "Uruguay": 1, "Paraguay": 1, "Argentina": 1, "Canada": 1, "Germany": 2` }}

</Example>

Table 20: Keyword counting for an example 4-passage split (GoT4): Execution plan (**GoO**)

GoO:

1. Split the input text into four paragraphs of roughly equal size (split_prompt)
2. For each paragraph: Count the occurrences of individual countries (count_prompt) 10 times; score each counting attempt; keep the best
3. Merge the country counts into one dictionary (merge_prompt) 3 times; validate and improve invalid merge attempts (improve_merge_prompt) up to 3 attempts each; score; keep the best

Table 21: Keyword counting for an example 4-passage split (GoT4): **Step 1 (Prompt/Response)**

Step 1 – Prompt:

<Instruction> Split the following input text into 4 paragraphs of approximately same length.

Only output the final 4 paragraphs in the following format without any additional text or thoughts:

```
{{
  "Paragraph 1": "Some paragraph text ...",
  "Paragraph 2": "Some paragraph text ...",
  "Paragraph 3": "Some paragraph text ...",
  "Paragraph 4": "Some paragraph text ..."
}}
```

</Instruction>

<Example> See Table 19 </Example>

Input: **My friend, Alex from Peru**, once recounted his journey to Argentina where he learned about the strong cultural ties between Argentina and Brazil due to their shared history. He spoke fondly of his time in Argentina and Brazil, marveling at the similar music, dance, and culinary traditions that seamlessly **bridged the borders of these countries. It reminded me of a** documentary I'd watched about the ancient Silk Road that spanned across Iran and China. It touched upon the shared historical narratives of Iran and China, highlighting how goods, ideas, and culture flowed between these two countries for centuries. Intriguingly, the documentary also brought up some parallels between this eastern route and the vibrant exchange between Italy and France during the Renaissance. The cultural and intellectual bonds between Italy and France were indeed profound, just as the resilient trade relations that existed **between the United States and Canada. The United States and Canada, apart** from their geographical proximity, shared an economic bond that strongly influenced their policies and international standing. Similarly, the ties between Australia and New Zealand shed light on their gestalt identity in the Pacific region. Despite their unique characteristics, Australia and New Zealand were often viewed as a single entity due to their remarkably similar backgrounds in terms of culture, language, and colonial history. Inspired by these chronicles of interconnectedness, I decided to delve deeper into history and found a fascinating account of how Ukraine and Poland had influenced each other **through years of coexistence. Despite their tumultuous past**, Ukraine and Poland shared a cultural tapestry that was deeply woven into their histories. It was quite similar to the complex relationship between North Korea and South Korea, which, despite their political differences, shared common heritage and traditions. Thus, from Argentina to South Korea, the world was an intricate web of countries intertwined with each other through shared histories, cultures, **and sometimes, shared destinies.**

Step 1 – Response (Fully Correct):

```
{{
  "Paragraph 1": "My friend, Alex from Peru ... (Omitted) ... bridged the borders of these countries.",
  "Paragraph 2": "It reminded me of a ... (Omitted) ... between the United States and Canada.",
  "Paragraph 3": "The United States and Canada, apart ... (Omitted) ... through years of coexistence.",
  "Paragraph 4": "Despite their tumultuous past ... (Omitted) ... and sometimes, shared destinies."
}}
```

Table 22: Keyword counting for an example 4-passage split (GoT4): **Step 2a (Prompt/Responses)**

Step 2a – Prompt:

<Instruction> Count the frequency of how many times each country is explicitly named in the input text. You can generate any intermedate lists and states, but the final output should only contain the frequency of each country that appears at least once in the following json format, prefixed with "Output: " (make sure to keep the same spelling for each country in the output as in the input text):

```
{{
  "country1": frequency1,
  "country2": frequency2,
  ...
}}
```

</Instruction>

<Approach>

To count the frequency for each country follow these steps:

1. Split the input passage into four paragraphs of similar length.
2. Count the frequency of each country in each paragraph.
3. Combine the frequencies of each country from each paragraph by adding them together.

</Approach>

<Examples> See Table 18 </Examples>

Input: My friend, Alex from Peru, once recounted his journey to Argentina where he learned about the strong cultural ties between Argentina and Brazil due to their shared history. He spoke fondly of his time in Argentina and Brazil, marveling at the similar music, dance, and culinary traditions that seamlessly bridged the borders of these countries.

Step 2a – 10 Responses:

1. `{{ "Peru": 1, "Argentina": 1, "Brazil": 1 }}` (3 Errors - Missing two "Argentina" and one "Brazil")
2. **`{{ "Peru": 1, "Argentina": 2, "Brazil": 2 }}` (1 Error - Missing one "Argentina")**
3. `{{ "Peru": 1, "Argentina": 2, "Brazil": 2 }}` (1 Error - Missing one "Argentina")
4. `{{ "Peru": 1, "Argentina": 1, "Brazil": 1 }}` (3 Errors - Missing two "Argentina" and one "Brazil")
5. `{{ "Peru": 1, "Argentina": 2, "Brazil": 2 }}` (1 Error - Missing one "Argentina")
6. `{{ "Peru": 1, "Argentina": 2, "Brazil": 2 }}` (1 Error - Missing one "Argentina")
7. `{{ "Peru": 1, "Argentina": 1, "Brazil": 1 }}` (3 Errors - Missing two "Argentina" and one "Brazil")
8. `{{ "Peru": 1, "Argentina": 1, "Brazil": 1 }}` (3 Errors - Missing two "Argentina" and one "Brazil")
9. `{{ "Peru": 1, "Argentina": 1, "Brazil": 1 }}` (3 Errors - Missing two "Argentina" and one "Brazil")
10. `{{ "Peru": 1, "Argentina": 1, "Brazil": 1 }}` (3 Errors - Missing two "Argentina" and one "Brazil")

Table 23: Keyword counting for an example 4-passage split (GoT4): **Step 2b (Prompt/Responses)**

Step 2b – Prompt:

<Instruction> Count the frequency of how many times each country is explicitly named in the input text. You can generate any intermedate lists and states, but the final output should only contain the frequency of each country that appears at least once in the following json format, prefixed with "Output: " (make sure to keep the same spelling for each country in the output as in the input text):

```
{{
  "country1": frequency1,
  "country2": frequency2,
  ...
}}
```

</Instruction>

<Approach>

To count the frequency for each country follow these steps:

1. Split the input passage into four paragraphs of similar length.
2. Count the frequency of each country in each paragraph.
3. Combine the frequencies of each country from each paragraph by adding them together.

</Approach>

<Examples> See Table 18 </Examples>

Input: It reminded me of a documentary I'd watched about the ancient Silk Road that spanned across Iran and China. It touched upon the shared historical narratives of Iran and China, highlighting how goods, ideas, and culture flowed between these two countries for centuries. Intriguingly, the documentary also brought up some parallels between this eastern route and the vibrant exchange between Italy and France during the Renaissance. The cultural and intellectual bonds between Italy and France were indeed profound, just as the resilient trade relations that existed between the United States and Canada.

Step 2b – 10 Responses:

1. `{{ "Iran": 1, "China": 1, "Italy": 1, "France": 1, "United States": 1, "Canada": 1 }}`
   (4 Errors - Missing one "Iran", one "China", one "Italy" and one "France")
2. `{{ "Iran": 1, "China": 1, "Italy": 1, "France": 1, "United States": 1, "Canada": 1 }}`
   (4 Errors - Missing one "Iran", one "China", one "Italy" and one "France")
3. `{{ "Iran": 3, "China": 2, "Italy": 1, "France": 1, "United States": 1, "Canada": 1 }}`
   (3 Errors - Surplus of one "Iran", missing one "Italy" and one "France")
4. `{{ "Iran": 1, "China": 1, "Italy": 1, "France": 1, "United States": 1, "Canada": 1 }}`
   (4 Errors - Missing one "Iran", one "China", one "Italy" and one "France")
5. `{{ "Iran": 2, "China": 2, "Italy": 1, "France": 1, "United States": 1, "Canada": 1 }}`
   (2 Errors - Missing one "Italy" and one "France")
6. `{{ "Iran": 1, "China": 1, "Italy": 1, "France": 1, "United States": 1, "Canada": 1 }}`
   (4 Errors - Missing one "Iran", one "China", one "Italy" and one "France")
7. `{{ "Iran": 1, "China": 1, "Italy": 1, "France": 1, "United States": 1, "Canada": 1 }}`
   (4 Errors - Missing one "Iran", one "China", one "Italy" and one "France")
8. `{{ "Iran": 1, "China": 1, "Italy": 1, "France": 1, "United States": 1, "Canada": 1 }}`
   (4 Errors - Missing one "Iran", one "China", one "Italy" and one "France")
9. `{{ "Iran": 1, "China": 1, "Italy": 1, "France": 1, "United States": 1, "Canada": 1 }}`
   (4 Errors - Missing one "Iran", one "China", one "Italy" and one "France")
10. **`{{ "Iran": 2, "China": 2, "Italy": 1, "France": 2, "United States": 1, "Canada": 1 }}`**
    (1 Error - Missing one "Italy")

Table 24: Keyword counting for an example 4-passage split (GoT4): **Step 2c (Prompt/Responses)**

Step 2c – Prompt:

<Instruction> Count the frequency of how many times each country is explicitly named in the input text. You can generate any intermedate lists and states, but the final output should only contain the frequency of each country that appears at least once in the following json format, prefixed with ”Output: ” (make sure to keep the same spelling for each country in the output as in the input text):

```
{{
  ”country1”: frequency1,
  ”country2”: frequency2,
  ...
}}
```

</Instruction>

<Approach>

To count the frequency for each country follow these steps:

1. Split the input passage into four paragraphs of similar length.
2. Count the frequency of each country in each paragraph.
3. Combine the frequencies of each country from each paragraph by adding them together.

</Approach>

<Examples> See Table 18 </Examples>

Input: The United States and Canada, apart from their geographical proximity, shared an economic bond that strongly influenced their policies and international standing. Similarly, the ties between Australia and New Zealand shed light on their gestalt identity in the Pacific region. Despite their unique characteristics, Australia and New Zealand were often viewed as a single entity due to their remarkably similar backgrounds in terms of culture, language, and colonial history. Inspired by these chronicles of interconnectedness, I decided to delve deeper into history and found a fascinating account of how Ukraine and Poland had influenced each other through years of coexistence.

Step 2c – 10 Responses:

1. **`{{ "United States": 1, "Canada": 1, "Australia": 2, "New Zealand": 2, "Ukraine": 1, "Poland": 1 }}`**
   (Fully Correct)
2. `{{ "United States": 1, "Canada": 1, "Australia": 2, "New Zealand": 2, "Ukraine": 1, "Poland": 1 }}`
   (Fully Correct)
3. `{{ "United States": 1, "Canada": 1, "Australia": 1, "New Zealand": 1, "Australia and New Zealand": 1, "Ukraine": 1, "Poland": 1 }}`
   (3 Errors - Missing one ”Australia, one ”New Zealand” and surplus of one ”Australia and New Zealand”)
4. `{{ "United States": 1, "Canada": 1, "Australia": 2, "New Zealand": 2, "Ukraine": 1, "Poland": 1 }}`
   (Fully Correct)
5. `{{ "United States": 1, "Canada": 1, "Australia": 1, "New Zealand": 1, "Australia and New Zealand": 1, "Ukraine": 1, "Poland": 1 }}`
   (3 Errors - Missing one ”Australia, one ”New Zealand” and surplus of one ”Australia and New Zealand”)
6. `{{ "United States": 1, "Canada": 1, "Australia": 1, "New Zealand": 1, "Ukraine": 1, "Poland": 1 }}`
   (2 Errors - Missing one ”Australia” and one ”New Zealand”)
7. `{{ "United States": 1, "Canada": 1, "Australia": 2, "New Zealand": 2, "Ukraine": 1, "Poland": 1 }}`
   (Fully Correct)
8. `{{ "United States": 1, "Canada": 1, "Australia": 1, "New Zealand": 1, "Ukraine": 1, "Poland": 1 }}`
   (2 Errors - Missing one ”Australia” and one ”New Zealand”)
9. `{{ "United States": 1, "Canada": 1, "Australia": 2, "New Zealand": 2, "Ukraine": 1, "Poland": 1 }}`
   (Fully Correct)
10. `{{ "United States": 1, "Canada": 1, "Australia": 2, "New Zealand": 2, "Ukraine": 1, "Poland": 1 }}`
    (Fully Correct)

Table 25: Keyword counting for an example 4-passage split (GoT4): **Step 2d (Prompt/Responses)**

Step 2d – Prompt:

<Instruction> Count the frequency of how many times each country is explicitly named in the input text. You can generate any intermedate lists and states, but the final output should only contain the frequency of each country that appears at least once in the following json format, prefixed with ”Output: ” (make sure to keep the same spelling for each country in the output as in the input text):

```
{{
  "country1": frequency1,
  "country2": frequency2,
  ...
}}
```

</Instruction>

<Approach>

To count the frequency for each country follow these steps:

1. Split the input passage into four paragraphs of similar length.
2. Count the frequency of each country in each paragraph.
3. Combine the frequencies of each country from each paragraph by adding them together.

</Approach>

<Examples> See Table 18 </Examples>

Input: Despite their tumultuous past, Ukraine and Poland shared a cultural tapestry that was deeply woven into their histories. It was quite similar to the complex relationship between North Korea and South Korea, which, despite their political differences, shared common heritage and traditions. Thus, from Argentina to South Korea, the world was an intricate web of countries intertwined with each other through shared histories, cultures, and sometimes, shared destinies.

Step 2d – 10 Responses:

1. **{{ "Ukraine": 1, "Poland": 1, "North Korea": 1, "South Korea": 2, "Argentina": 1 }} (Fully Correct)**
2. {{ "Ukraine": 1, "Poland": 1, "North Korea": 1, "South Korea": 2, "Argentina": 1 }} (Fully Correct)
3. {{ "Ukraine": 1, "Poland": 1, "North Korea": 1, "South Korea": 2, "Argentina": 1 }} (Fully Correct)
4. {{ "Ukraine": 1, "Poland": 1, "North Korea": 1, "South Korea": 2, "Argentina": 1 }} (Fully Correct)
5. {{ "Ukraine": 1, "Poland": 1, "North Korea": 1, "South Korea": 2, "Argentina": 1 }} (Fully Correct)
6. {{ "Ukraine": 1, "Poland": 1, "North Korea": 1, "South Korea": 2, "Argentina": 1 }} (Fully Correct)
7. {{ "Ukraine": 1, "Poland": 1, "North Korea": 1, "South Korea": 2, "Argentina": 1 }} (Fully Correct)
8. {{ "Ukraine": 1, "Poland": 1, "North Korea": 1, "South Korea": 2, "Argentina": 1 }} (Fully Correct)
9. {{ "Ukraine": 1, "Poland": 1, "North Korea": 1, "South Korea": 2, "Argentina": 1 }} (Fully Correct)
10. {{ "Ukraine": 1, "Poland": 1, "North Korea": 1, "South Korea": 2, "Argentina": 1 }} (Fully Correct)

Table 26: Keyword counting for an example 4-passage split (GoT4): **Step 3a (Prompt/Responses)**

Step 3a – Prompt:

<Instruction> Combine the following 2 dictionaries, each containing the frequency of countries in a text, into a single dictionary. Simply add the frequencies together for each country and if a country is not present in one of the dictionaries, add it to the final dictionary with the frequency from the other dictionary.

Only output the final merged dictionary without any additional text or thoughts! </Instruction>

<Approach>

To combine the 2 dictionaries into single one, follow these steps:

1. Create a new dictionary to store the combined frequencies.
2. Iterate through the keys of the first dictionary and add the frequency of each country to the new dictionary.
3. Iterate through the keys of the second dictionary and add the frequency of each country to the new dictionary and if it is already present, add the frequency to the existing value.

</Approach>

Combine the following 2 dictionaries into a single dictionary:

`{{ "Peru": 1, "Argentina": 2, "Brazil": 2 }}`

`{{ "Iran": 2, "China": 2, "Italy": 1, "France": 2, "United States": 1, "Canada": 1 }}`

Combined Output:

Step 3a – 3 Responses:

1. **`{{ "Peru": 1, "Argentina": 2, "Brazil": 2, "Iran": 2, "China": 2, "Italy": 1, "France": 2, "United States": 1, "Canada": 1 }}` (Valid - No Improvement) (2 Errors - from counting)**
2. `{{ "Peru": 1, "Argentina": 2, "Brazil": 2, "Iran": 2, "China": 2, "Italy": 1, "France": 2, "United States": 1, "Canada": 1 }}` (Valid - No Improvement) (2 Errors - from counting)
3. `{{ "Peru": 1, "Argentina": 2, "Brazil": 2, "Iran": 2, "China": 2, "Italy": 1, "France": 2, "United States": 1, "Canada": 1 }}` (Valid - No Improvement) (2 Errors - from counting)

Table 27: Keyword counting for an example 4-passage split (GoT4): **Step 3b (Prompt/Responses)**

Step 3b – Prompt:

<Instruction> Combine the following 2 dictionaries, each containing the frequency of countries in a text, into a single dictionary. Simply add the frequencies together for each country and if a country is not present in one of the dictionaries, add it to the final dictionary with the frequency from the other dictionary.

Only output the final merged dictionary without any additional text or thoughts! </Instruction>

<Approach>

To combine the 2 dictionaries into single one, follow these steps:

1. Create a new dictionary to store the combined frequencies.
2. Iterate through the keys of the first dictionary and add the frequency of each country to the new dictionary.
3. Iterate through the keys of the second dictionary and add the frequency of each country to the new dictionary and if it is already present, add the frequency to the existing value.

</Approach>

Combine the following 2 dictionaries into a single dictionary:

`{{ "United States": 1, "Canada": 1, "Australia": 2, "New Zealand": 2, "Ukraine": 1, "Poland": 1 }}`

`{{ "Ukraine": 1, "Poland": 1, "North Korea": 1, "South Korea": 2, "Argentina": 1 }}`

Combined Output:

Step 3b – 3 Responses:

1. **`{{ "United States": 1, "Canada": 1, "Australia": 2, "New Zealand": 2, "Ukraine": 2, "Poland": 2, "North Korea": 1, "South Korea": 2, "Argentina": 1 }}`** **(Valid - No Improvement) (Fully Correct)**
2. `{{ "United States": 1, "Canada": 1, "Australia": 2, "New Zealand": 2, "Ukraine": 2, "Poland": 2, "North Korea": 1, "South Korea": 2, "Argentina": 1 }}` (Valid - No Improvement) (Fully Correct)
3. `{{ "United States": 1, "Canada": 1, "Australia": 2, "New Zealand": 2, "Ukraine": 2, "Poland": 2, "North Korea": 1, "South Korea": 2, "Argentina": 1 }}` (Valid - No Improvement) (Fully Correct)

Table 28: Keyword counting for an example 4-passage split (GoT4): **Step 3c (Prompt/Responses)**

Step 3c – Prompt:

<Instruction> Combine the following 2 dictionaries, each containing the frequency of countries in a text, into a single dictionary. Simply add the frequencies together for each country and if a country is not present in one of the dictionaries, add it to the final dictionary with the frequency from the other dictionary.

Only output the final merged dictionary without any additional text or thoughts! </Instruction>

<Approach>

To combine the 2 dictionaries into single one, follow these steps:

1. Create a new dictionary to store the combined frequencies.
2. Iterate through the keys of the first dictionary and add the frequency of each country to the new dictionary.
3. Iterate through the keys of the second dictionary and add the frequency of each country to the new dictionary and if it is already present, add the frequency to the existing value.

</Approach>

Combine the following 2 dictionaries into a single dictionary:

`{{ "Peru": 1, "Argentina": 2, "Brazil": 2, "Iran": 2, "China": 2, "Italy": 1, "France": 2, "United States": 1, "Canada": 1 }}`

`{{ "United States": 1, "Canada": 1, "Australia": 2, "New Zealand": 2, "Ukraine": 2, "Poland": 2, "North Korea": 1, "South Korea": 2, "Argentina": 1 }}`

Combined Output:

Step 3c – 3 Responses:

1. **`{{ "Peru": 1, "Argentina": 3, "Brazil": 2, "Iran": 2, "China": 2, "Italy": 1, "France": 2, "United States": 2, "Canada": 2, "Australia": 2, "New Zealand": 2, "Ukraine": 2, "Poland": 2, "North Korea": 1, "South Korea": 2 }}` (Valid - No Improvement) (2 Errors - from counting)**
2. `{{ "Peru": 1, "Argentina": 3, "Brazil": 2, "Iran": 2, "China": 2, "Italy": 1, "France": 2, "United States": 2, "Canada": 2, "Australia": 2, "New Zealand": 2, "Ukraine": 2, "Poland": 2, "North Korea": 1, "South Korea": 2 }}` (Valid - No Improvement) (2 Errors - from counting)
3. `{{ "Peru": 1, "Argentina": 3, "Brazil": 2, "Iran": 2, "China": 2, "Italy": 1, "France": 2, "United States": 2, "Canada": 2, "Australia": 2, "New Zealand": 2, "Ukraine": 2, "Poland": 2, "North Korea": 1, "South Korea": 2 }}` (Valid - No Improvement) (2 Errors - from counting)

Final Result (2 Errors):

`{{ "Peru": 1, "Argentina": 3, "Brazil": 2, "Iran": 2, "China": 2, "Italy": 1, "France": 2, "United States": 2, "Canada": 2, "Australia": 2, "New Zealand": 2, "Ukraine": 2, "Poland": 2, "North Korea": 1, "South Korea": 2 }}`

E Example Prompts - Document Merging

We present the prompts only for `GoT` of the document merging task, as `GoT2` only differs in the fact that it merges the 4 NDAs in 2 steps rather than 1. For document merging, we employ four distinct types of operations that interact with the LLM, each with its corresponding prompts. First, there is the **Generate** operation, utilizing the *merge_prompt* to instruct the LLM to merge the 4 NDAs into 1. Second, the **Score** operations instructs the LLM to score a given merged NDA using the *score_prompt*. Next, the **Aggregate** operation employs the *aggregate_prompt* to instruct the LLM to aggregate multiple merge attempts into a single, better one. Finally, the **Improve** operation leverages the *improve_prompt* to instruct the LLM to improve a merged NDA.

First, we present the prompt stubs (Table 29 - Table 30), serving as templates to dynamically generate appropriate prompts at runtime. Following this, we outline the LLM interactions throughout a complete merging process (Table 31 - Table 49). However, instead of displaying each input/generated NDA in every prompt/response, we present the 4 input NDAs in Table 31 - Table 33 and the final merged NDA in Table 49. Furthermore, as scoring is done using the LLM as well, we will present these interactions for the best performing merged NDAs (Tables 39 - 40 and Tables 47 - 48). Lastly, most responses are limited to a few lines only, as they don't offer any further insights and would otherwise span multiple pages. However, we refer the interested reader to the results in the corresponding code repository[2] for full logs and further examples.

---

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

Table 29: Prompt stubs for the document merging task; parameters in single curly brackets will be substituted at runtime.

---

merge_prompt: Merge the following 4 NDA documents <Doc1> - <Doc4> into a single NDA, maximizing retained information and minimizing redundancy. Output only the created NDA between the tags <Merged> and </Merged>, without any additional text.

Here are NDAs <Doc1> - <Doc4>:

<Doc1> {doc1} </Doc1>

<Doc2> {doc2} </Doc2>

<Doc3> {doc3} </Doc3>

<Doc4> {doc4} </Doc4>

---

score_prompt: The following NDA <S> merges NDAs <Doc1> - <Doc4>.

Please score the merged NDA <S> in terms of how much redundant information is contained, independent of the original NDAs, as well as how much information is retained from the original NDAs.

A score of 10 for redundancy implies that absolutely no information is redundant, while a score of 0 implies that at least half of the information is redundant (so everything is at least mentioned twice).

A score of 10 for retained information implies that all information from the original NDAs is retained, while a score of 0 implies that no information is retained.

You may provide reasoning for your scoring, but the final score for redundancy should be between the tags <Redundancy> and </Redundancy>, and the final score for retained information should be between the tags <Retained> and </Retained>, without any additional text within any of those tags.

Here are NDAs <Doc1> - <Doc4>:

<Doc1> {doc1} </Doc1>

<Doc2> {doc2} </Doc2>

<Doc3> {doc3} </Doc3>

<Doc4> {doc4} </Doc4>

Here is the merged NDA <S>:

<S> {s} </S>

---

aggregate_prompt: The following NDAs <S1> - <S{num_ndas_summaries}> each merge the initial NDAs <Doc1> - <Doc4>.

Combine the merged NDAs <S1> - <S{num_ndas_summaries}> into a new one, maximizing their advantages and overall information retention, while minimizing redundancy.

Output only the new NDA between the tags <Merged> and </Merged>, without any additional text.

Here are the original NDAs <Doc1> - <Doc4>:

<Doc1> {doc1} </Doc1>

<Doc2> {doc2} </Doc2>

<Doc3> {doc3} </Doc3>

<Doc4> {doc4} </Doc4>

Here are the merged NDAs <S1> - <S{num_ndas_summaries}>:

<S1> {s1} </S1>

. . .

<S{num_ndas_summaries}> {s{num_ndas_summaries}} </S{num_ndas_summaries}>

Table 30: Prompt stubs for the document merging task *continued*; parameters in single curly brackets will be substituted at runtime.

---

improve_prompt: The following NDA <S> merges initial NDAs <Doc1> - <Doc4>.

Please improve the merged NDA <S> by adding more information and removing redundancy. Output only the improved NDA, placed between the tags <Merged> and </Merged>, without any additional text.

Here are NDAs <Doc1> - <Doc4>:

<Doc1> {doc1} </Doc1>

<Doc2> {doc2} </Doc2>

<Doc3> {doc3} </Doc3>

<Doc4> {doc4} </Doc4>

Here is the merged NDA <S>:

<S> {s} </S>

Table 31: Input NDA 1 and 2

<Doc1>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].
3. ”Confidential Information” includes all potentially commercially valuable information, specifically software development tactics, processes, and in-house research results.
4. Receiving party is obligated to protect the Confidential Information, use it solely for the disclosed purpose, and not disclose it without consent.
5. Breach penalties include injunctive relief, other remedies, and a $200,000 fee per breach.
6. The Agreement applies to the Parties and their successors and assigns. It contains all related agreements and lack of enforcement doesn’t imply waiver.
7. The Agreement is under the laws of [State].
8. Signed by [Your Company Name] and [Recipient Name] at the above date.

</Doc1>

<Doc2>

NON-DISCLOSURE AGREEMENT (NDA)

Effective from [Effective Date], this NDA involves [Your Company Name] (”Disclosing Party”), and [Recipient Name] (”Receiving Party”).

1. Purpose: The Disclosing Party will disclose confidential information related to [Topic of Research] to the Receiving Party for [Purpose].
2. Confidential Information: Defined as all non-public reports, data, designs, and other materials provided by the Disclosing Party to the Receiving Party.
3. Receiving Party’s Obligations:
   a. Use, reproduce, or distribute the confidential information only for the agreed purpose.
   b. Restrict access to the information to necessary parties, ensuring they abide by strict confidentiality.
   c. Return or destroy all confidential information upon request or at the end of the agreement.
4. Exclusions: Information will not be classified as confidential if it is already known to the Receiving Party, publicly known, or independently developed by the Receiving Party.
5. Non-Competition: The Receiving Party will not engage in any competing business against the Disclosing Party during the agreement and one year after its termination.
6. Term and Termination: The agreement is valid for [e.g., ”two years”], unless terminated earlier with [e.g., ”30 days”] written notice. The Receiving Party’s non-disclosure and non-competition obligations persist post-termination.
7. General Provisions:
   a. Governing Law: [Your State]’s laws apply.
   b. Amendments: Only valid if written and signed by both parties.
   c. Entire Agreement: This contract overrules previous related agreements.

Signed as of the Effective Date by [Your Company Name] - Disclosing Party [Recipient Name] - Receiving Party.

</Doc2>

Table 32: Input NDA 3

<Doc3>

CONFIDENTIALITY & NON-DISCLOSURE AGREEMENT

Entities Involved:

Effective [Date], between [AquaBlue Innovations], established in [State], and [PineTree Solutions], a registered entity.

Objective:

To safeguard classified data during talks of a potential technological alliance.

Specification of Protected Information:

Particularly:

a. System designs and architectural schematics.
b. Proprietary computational algorithms.

Receiver's Obligations:

a. Maintain strict non-disclosure using best practices.
b. Employ solely for the aforementioned aim.
c. No unveiling without explicit authorization.

Violation Ramifications:

A charge of $280,000 for every infringement, plus possible legal proceedings.

General Terms:

Binding for both parties and any successors. This encapsulates the entire accord.

Legal Reference:

Governed as per [State]'s legal framework.

Attestation:

Duly signed on [Date].
[AquaBlue Innovations] [PineTree Solutions]

</Doc3>

Table 33: Input NDA 4

<Doc4>

SECRECY & DISCLOSURE AGREEMENT

Contracting Parties:

Dated [Date], drawn between [AquaBlue Innovations], a [State]-based corporation, and [PineTree Solutions], a licensed organization.

Aim:

To protect exclusive insights amidst dialogues for a technological partnership.

Categorization of Sensitive Data:

Includes:

a. Internal software blueprints.
b. Intellectual property awaiting patents.

Commitments of Recipient:

a. Uphold confidentiality, ensuring data integrity.
b. Utilize strictly for collaborative ventures.
c. No exposure without prior consensus.

Repercussions for Non-Compliance:

$295,000 fine for each transgression, and the option for legal recourse.

Overall Provisions:

Legally enforceable for signatories and successors. Complete and sole agreement.

Juridical Standpoint:

Under the auspices of [State] laws.

Ratification:

Confirmed and endorsed on [Date].
[AquaBlue Innovations] [PineTree Solutions]

</Doc4>

Table 34: Merging 4 example NDAs: Execution plan (**GoO**)

GoO:

1. Merge the 4 NDAs into a single one 5 times; Score each attempt and keep the best 3
2. Aggregate the merge attempts into a single one 5 times; Score each aggregation attempt and keep the overall best attempt (including Step 1)
3. Improve the merged NDA 10 times; Score each and keep the best

Table 35: Merging 4 example NDAs: **Step 1 (Prompt)**

Merge the following 4 NDA documents <Doc1> - <Doc4> into a single NDA, maximizing retained information and minimizing redundancy. Output only the created NDA between the tags <Merged> and </Merged>, without any additional text.

Here are NDAs <Doc1> - <Doc4>:

<Doc1>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted, see Table 31)

</Doc1>

<Doc2>

NON-DISCLOSURE AGREEMENT (NDA)

Effective from [Effective Date], this NDA involves [Your Company Name] ("Disclosing Party"), and [Recipient Name] ("Receiving Party").

1. Purpose: The Disclosing Party will disclose confidential information related to [Topic of Research] to the Receiving Party for [Purpose].

. . . (Omitted, see Table 31)

</Doc2>

<Doc3>

CONFIDENTIALITY & NON-DISCLOSURE AGREEMENT

Entities Involved:

Effective [Date], between [AquaBlue Innovations], established in [State], and [PineTree Solutions], a registered entity.

Objective:

To safeguard classified data during talks of a potential . . . (Omitted, see Table 32)

</Doc3>

<Doc4>

SECRECY & DISCLOSURE AGREEMENT

Contracting Parties:

Dated [Date], drawn between [AquaBlue Innovations], a [State]-based corporation, and [PineTree Solutions], a licensed organization.

Aim:

To protect exclusive insights amidst . . . (Omitted, see Table 33)

</Doc4>

Table 36: Merging 4 example NDAs; we only present the highest scoring response fully in Tables 37-38: **Step 1 (Responses)**

Note that due to the shortening of responses, the appear to be equal, but small differences can be found when observing the fully generated summaries. The only small differences are reflected in the very similar scores (see Table 39-40 for the scoring of the best response).

Response (1/5):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted)

Score: 6.60

Response (2/5):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted, see Tables 37 - 38)

Score: 6.87

Response (3/5):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted)

Score: 6.60

Response (4/5):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted)

Score: 5.78

Response (5/5):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted)

Score: 6.50

Table 37: Merging 4 example NDAs: **Step 1 (Highest Scoring Response Part 1)**

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].
3. ”Confidential Information” includes all potentially commercially valuable information, specifically software development tactics, processes, and in-house research results.
4. Receiving party is obligated to protect the Confidential Information, use it solely for the disclosed purpose, and not disclose it without consent.
5. Breach penalties include injunctive relief, other remedies, and a $200,000 fee per breach.
6. The Agreement applies to the Parties and their successors and assigns. It contains all related agreements and lack of enforcement doesn’t imply waiver.
7. The Agreement is under the laws of [State].
8. Signed by [Your Company Name] and [Recipient Name] at the above date.

Effective from [Effective Date], this NDA involves [Your Company Name] (”Disclosing Party”), and [Recipient Name] (”Receiving Party”).

Purpose: The Disclosing Party will disclose confidential information related to [Topic of Research] to the Receiving Party for [Purpose].

Confidential Information: Defined as all non-public reports, data, designs, and other materials provided by the Disclosing Party to the Receiving Party.

Receiving Party’s Obligations:

a. Use, reproduce, or distribute the confidential information only for the agreed purpose.
b. Restrict access to the information to necessary parties, ensuring they abide by strict confidentiality.
c. Return or destroy all confidential information upon request or at the end of the agreement.

Exclusions: Information will not be classified as confidential if it is already known to the Receiving Party, publicly known, or independently developed by the Receiving Party.

Non-Competition: The Receiving Party will not engage in any competing business against the Disclosing Party during the agreement and one year after its termination.

Term and Termination: The agreement is valid for [e.g., ”two years”], unless terminated earlier with [e.g., ”30 days”] written notice. The Receiving Party’s non-disclosure and non-competition obligations persist post-termination.

General Provisions:

a. Governing Law: [Your State]’s laws apply.
b. Amendments: Only valid if written and signed by both parties.
c. Entire Agreement: This contract overrules previous related agreements.

Signed as of the Effective Date by [Your Company Name] - Disclosing Party [Recipient Name] - Receiving Party.

Entities Involved:

Effective [Date], between [AquaBlue Innovations], established in [State], and [PineTree Solutions], a registered entity.

Objective:

To safeguard classified data during talks of a potential technological alliance.

Table 38: Merging 4 example NDAs: **Step 1 (Highest Scoring Response Part 2)**

Specification of Protected Information:

Particularly:

a. System designs and architectural schematics.
b. Proprietary computational algorithms.

Receiver’s Obligations:

a. Maintain strict non-disclosure using best practices.
b. Employ solely for the aforementioned aim.
c. No unveiling without explicit authorization.

Violation Ramifications:

A charge of $280,000 for every infringement, plus possible legal proceedings.

General Terms:

Binding for both parties and any successors. This encapsulates the entire accord.

Legal Reference:

Governed as per [State]’s legal framework.

Attestation:

Duly signed on [Date].

Contracting Parties:

Dated [Date], drawn between [AquaBlue Innovations], a [State]-based corporation, and [PineTree Solutions], a licensed organization.

Aim:

To protect exclusive insights amidst dialogues for a technological partnership.

Categorization of Sensitive Data:

Includes:

a. Internal software blueprints.
b. Intellectual property awaiting patents.

Commitments of Recipient:

a. Uphold confidentiality, ensuring data integrity.
b. Utilize strictly for collaborative ventures.
c. No exposure without prior consensus.

Repercussions for Non-Compliance:

$295,000 fine for each transgression, and the option for legal recourse.

Overall Provisions:

Legally enforceable for signatories and successors. Complete and sole agreement.

Juridical Standpoint:

Under the auspices of [State] laws.

Ratification:

Confirmed and endorsed on [Date].

Table 39: Merging 4 example NDAs: **Step 1 (Scoring Example – Prompt)**

Please score the merged NDA <S> in terms of how much redundant information is contained, independent of the original NDAs, as well as how much information is retained from the original NDAs.

A score of 10 for redundancy implies that absolutely no information is redundant, while a score of 0 implies that at least half of the information is redundant (so everything is at least mentioned twice).

A score of 10 for retained information implies that all information from the original NDAs is retained, while a score of 0 implies that no information is retained.

You may provide reasoning for your scoring, but the final score for redundancy should be between the tags <Redundancy> and </Redundancy>, and the final score for retained information should be between the tags <Retained> and </Retained>, without any additional text within any of those tags.

Here are NDAs <Doc1> - <Doc4>:

<Doc1>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].

. . . (Omitted, see Table 31)

</Doc1>

<Doc2>

NON-DISCLOSURE AGREEMENT (NDA)

Effective from [Effective Date], this NDA involves [Your Company Name] (”Disclosing Party”), and [Recipient Name] (”Receiving Party”).

. . . (Omitted, see Table 31)

</Doc2>

<Doc3>

CONFIDENTIALITY & NON-DISCLOSURE AGREEMENT

Entities Involved:

Effective [Date], between [AquaBlue Innovations], established in [State], and . . . (Omitted, see Table 32)

</Doc3>

<Doc4>

SECRECY & DISCLOSURE AGREEMENT

Contracting Parties:

Dated [Date], drawn between [AquaBlue Innovations], a [State]-based corporation, and . . . (Omitted, see Table 33)

</Doc4>

Here is the summary NDA <S>:

<S>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted, see Tables 37-38)

</S>

Table 40: Merging 4 example NDAs: **Step 1 (Scoring Example – Responses)**

Response (1/3):

<Redundancy>5</Redundancy>

<Retained>10</Retained>

Reasoning:

In terms of redundancy, there is some overlap in the information contained in the NDAs, such as the definition of ”Confidential Information” and the obligations of the receiving party. However, there are also some unique details in each NDA, such as the specific purpose of information sharing and the penalties for breach. Overall, there is a moderate level of redundancy in the merged NDA.

In terms of retained information, all the key elements from the original NDAs are included in the merged NDA. This includes the agreement between the parties, the purpose of information sharing, the definition of confidential information, the obligations of the receiving party, the exclusions for confidential information, non-competition clauses, the term and termination of the agreement, and the general provisions. Therefore, the merged NDA retains all the important information from the original NDAs.

Response (2/3):

<Redundancy>8</Redundancy>

<Retained>10</Retained>

Reasoning:

The merged NDA <S> contains multiple redundant sections from the original NDAs. For example, the introductory sections, such as the agreement between the parties and the purpose of information sharing, are repeated multiple times. Additionally, sections related to the definition of confidential information, the recipient’s obligations, exclusions, non-competition, term and termination, and general provisions are also redundantly included.

However, in terms of retained information, the merged NDA retains all the key elements from the original NDAs, including the parties involved, the objective, specification of protected information, recipient’s obligations, violation ramifications, general terms, legal reference, and attestation. Overall, while there is redundancy in the merged NDA, it successfully retains all the necessary information from the original NDAs.

Response (3/3):

<Redundancy>3</Redundancy>

The merged NDA contains some redundancy. There are three sections that have overlapping information between the original NDAs. These sections include:

1. The agreement between the parties (Section 1 in Doc1 and the beginning of Doc2)
2. Information sharing and the purpose of the NDA (Section 2 in both Doc1 and Doc2)
3. General provisions such as governing law and amendments (Section 7 in Doc1 and Doc2)

<Retained>9</Retained>

The merged NDA retains a significant amount of information from the original NDAs. It includes all key clauses and provisions, such as the definition of confidential information, the obligations of the receiving party, exclusions, non-competition, and general provisions. Only a few minor details, such as specific penalties and fine amounts, are not retained in the merged NDA. Overall, the merged NDA effectively combines the essential elements from the original NDAs.

Note: Remember to remove the reasoning from between the tags when providing the final scores.

Final Overall Score (Harmonic Mean of Averages):

6.87

Table 41: Merging 4 example NDAs; Note that S1-S3 are the three best scoring merge attemps from the previous step: **Step 2 (Prompt)**

The following NDAs <S1> - <S3> each merge the initial NDAs <Doc1> - <Doc4>.
Combine the merged NDAs <S1> - <S3> into a new one, maximizing their advantages and overall information retention, while minimizing redundancy.
Output only the new NDA between the tags <Merged> and </Merged>, without any additional text.

Here are the original NDAs <Doc1> - <Doc4>:

<Doc1>

. . . (Omitted, see Table 31)

</Doc1>

<Doc2>

. . . (Omitted, see Table 31)

</Doc2>

<Doc3>

. . . (Omitted, see Table 32)

</Doc3>

<Doc4>

. . . (Omitted, see Table 33)

</Doc4>

Here are the merged NDAs <S1> - <S3>:

<S1>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].

. . . (Omitted)

</S1>

<S2>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].

. . . (Omitted, see Tables 37-38)

</S2>

<S3>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].

. . . (Omitted)

</S3>

Table 42: Merging 4 example NDAs: **Step 2 (Responses)**

For this run, none of the responses improve upon the previous best and thus will all be discarded by the next step. For that reason and due to their long length, we do not present them fully and exclude their scoring.

Response (1/5):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted)

Score: 6.50

Response (2/5):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

Effective from [Effective Date], this NDA involves [Your Company Name] (”Disclosing Party”), and [Recipient Name] (”Receiving Party”).

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].

. . . (Omitted)

Score: 5.62

Response (3/5):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted)

Score: 6.60

Response (4/5):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted)

Score: 6.15

Response (5/5):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted)

Score: 6.22

Table 43: Merging 4 example NDAs: **Step 3 (Prompt)**

The following NDA <S> merges initial NDAs <Doc1> - <Doc4>.

Please improve the merged NDA <S> by adding more information and removing redundancy. Output only the improved NDA, placed between the tags <Merged> and </Merged>, without any additional text.

Here are NDAs <Doc1> - <Doc4>:

<Doc1>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted, see Table 31)

</Doc1>

<Doc2>

NON-DISCLOSURE AGREEMENT (NDA)

Effective from [Effective Date], this NDA involves [Your Company Name] (”Disclosing Party”), and [Recipient Name] (”Receiving Party”).

1. Purpose: The Disclosing Party will disclose confidential information related to [Topic of Research] to the Receiving Party for [Purpose].

. . . (Omitted, see Table 31)

</Doc2>

<Doc3>

CONFIDENTIALITY & NON-DISCLOSURE AGREEMENT

Entities Involved:

Effective [Date], between [AquaBlue Innovations], established in [State], and [PineTree Solutions], a registered entity.

Objective:

To safeguard classified data during talks of a potential . . . (Omitted, see Table 32)

</Doc3>

<Doc4>

SECRECY & DISCLOSURE AGREEMENT

Contracting Parties:

Dated [Date], drawn between [AquaBlue Innovations], a [State]-based corporation, and [PineTree Solutions], a licensed organization.

Aim:

To protect exclusive insights amidst . . . (Omitted, see Table 33)

</Doc4>

Here is the summary NDA <S>:

<S>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted, see Tables 37-38)

</S>

Table 44: Merging 4 example NDAs; we only present the highest scoring response fully in Table 46: **Step 3 (Responses Part 1)**

Response (1/10):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted)

Score: 6.35

Response (2/10):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted)

Score: 6.35

Response (3/10):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date], for the purpose of [specific project or purpose].
2. ”Confidential Information” includes all potentially commercially valuable information, specifically software development tactics, processes, in-house research results, system designs, architectural schematics, proprietary computational algorithms, internal software blueprints, and intellectual property awaiting patents.

. . . (Omitted)

Score: 6.87

Response (4/10):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date] for the purpose of [specific project or purpose].
2. ”Confidential Information” includes all potentially commercially valuable information, such as software development tactics, processes, in-house research results, system designs, architectural schematics, proprietary computational algorithms, internal software blueprints, and intellectual property awaiting patents.

. . . (Omitted, see Table 46))

Score: 7.78

Response (5/10):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted)

Score: 6.43

Table 45: Merging 4 example NDAs; we only present the highest scoring response fully in Table 46: **Step 3 (Responses Part 2)**

Response (6/10):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose].

. . . (Omitted)

Score: 6.35

Response (7/10):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date] for the purpose of [specific project or purpose].
2. ”Confidential Information” includes all potentially commercially valuable information, specifically software development tactics, processes, in-house research results, system designs, architectural schematics, proprietary computational algorithms, internal software blueprints, and intellectual property awaiting patents.

. . . (Omitted)

Score: 6.70

Response (8/10):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date] for the purpose of [specific project or purpose].
2. ”Confidential Information” includes potentially commercially valuable information, such as software development tactics, processes, in-house research results, non-public reports, data, designs, and other materials provided by the Disclosing Party.

. . . (Omitted)

Score: 5.33

Response (9/10):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].
2. Information sharing for the purpose of [specific project or purpose], including [Topic of Research].

. . . (Omitted)

Score: 7.30

Response (10/10):

<Merged>

NON-DISCLOSURE AGREEMENT (NDA)

Agreement between [Your Company Name] and [Recipient Name] on [Date].

Information sharing for the purpose of [specific project or purpose].

. . . (Omitted)

Score: 6.40

Table 46: Merging 4 example NDAs: **Step 3 (Highest Scoring Response)**

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date] for the purpose of [specific project or purpose].
2. ”Confidential Information” includes all potentially commercially valuable information, such as software development tactics, processes, in-house research results, system designs, architectural schematics, proprietary computational algorithms, internal software blueprints, and intellectual property awaiting patents.
3. Receiving party is obligated to protect the Confidential Information, use it solely for the disclosed purpose, and not disclose it without consent.
4. Breach penalties include injunctive relief, other remedies, and a fine per breach (e.g., $200,000 or $280,000 for each infringement, depending on the agreement).
5. The Agreement applies to the Parties and their successors and assigns. It contains all related agreements and lack of enforcement doesn’t imply waiver.
6. The Agreement is under the laws of [State] and is valid for a specified period (e.g., two years) unless terminated earlier with written notice (e.g., 30 days).
7. The Receiving Party agrees not to engage in any competing business against the Disclosing Party during the agreement and a specified period after its termination (e.g., one year).
8. The Agreement can only be amended if written and signed by both parties, and overrules previous related agreements.
9. The Agreement is legally enforceable and binding for signatories and successors.
10. The Agreement is governed by the laws of [State].

Signed as of the Effective Date by [Your Company Name] - Disclosing Party [Recipient Name] - Receiving Party, along with the other parties involved.

Table 47: Merging 4 example NDAs: **Step 3 (Scoring Example – Prompt)**

Please score the merged NDA <S> in terms of how much redundant information is contained, independent of the original NDAs, as well as how much information is retained from the original NDAs.

A score of 10 for redundancy implies that absolutely no information is redundant, while a score of 0 implies that at least half of the information is redundant (so everything is at least mentioned twice).

A score of 10 for retained information implies that all information from the original NDAs is retained, while a score of 0 implies that no information is retained.

You may provide reasoning for your scoring, but the final score for redundancy should be between the tags <Redundancy> and </Redundancy>, and the final score for retained information should be between the tags <Retained> and </Retained>, without any additional text within any of those tags.

Here are NDAs <Doc1> - <Doc4>:

<Doc1>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date].

. . . (Omitted, see Table 31)

</Doc1>

<Doc2>

NON-DISCLOSURE AGREEMENT (NDA)

Effective from [Effective Date], this NDA involves [Your Company Name] ("Disclosing Party"), and [Recipient Name] ("Receiving Party").

. . . (Omitted, see Table 31)

</Doc2>

<Doc3>

CONFIDENTIALITY & NON-DISCLOSURE AGREEMENT

Entities Involved:

Effective [Date], between [AquaBlue Innovations], established in [State], and . . . (Omitted, see Table 32)

</Doc3>

<Doc4>

SECRECY & DISCLOSURE AGREEMENT

Contracting Parties:

Dated [Date], drawn between [AquaBlue Innovations], a [State]-based corporation, and . . . (Omitted, see Table 33)

</Doc4>

Here is the summary NDA <S>:

<S>

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date] for the purpose of [specific project or purpose].
2. "Confidential Information" includes all potentially commercially valuable information, such as software development tactics, processes, in-house research results, system designs, architectural schematics, proprietary computational algorithms, internal software blueprints, and intellectual property awaiting patents.

. . . (Omitted, see Table 46)

</S>

Table 48: Merging 4 example NDAs: **Step 3 (Scoring Example – Responses)**

Response (1/3):

<Redundancy>5</Redundancy>

<Retained>8</Retained>

Reasoning:

The merged NDA <S> retains most of the information from the original NDAs, combining relevant clauses and terms into a single comprehensive agreement. However, some redundancy is present as similar information is mentioned multiple times, such as the definition of ”Confidential Information” and the obligations of the receiving party. Additionally, some specific details from the original NDAs are not included in the merged NDA, such as the exclusion criteria in Doc2 and the provisions for non-competition in Doc3 and Doc4. Nonetheless, the merged NDA captures the essence of the original NDAs and includes all key information necessary for a non-disclosure agreement.

Response (2/3):

<Redundancy>8</Redundancy>

<Retained>10</Retained>

Reasoning:

The merged NDA <S> retains all the key information from the original NDAs, including the purpose of the agreement, the definition of confidential information, the obligations of the receiving party, breach penalties, applicability to parties and successors, governing law, and signature requirements.

There is some redundant information, such as the inclusion of system designs, architectural schematics, proprietary computational algorithms, internal software blueprints, and intellectual property awaiting patents from both Doc3 and Doc4. However, this redundancy is minor and does not significantly impact the overall usefulness of the NDA.

Overall, the merged NDA <S> retains all important information while minimizing redundancy, making it highly effective in conveying the necessary terms and obligations of the agreement. Therefore, it receives a score of 8 for redundancy and a score of 10 for retained information.

Response (3/3):

<Redundancy>7</Redundancy>

<Retained>10</Retained>

Reasoning:

The merged NDA <S> contains some redundant information, such as the definition of ”Confidential Information” and the obligations of the receiving party. However, it retains all the important information from the original NDAs, including the purpose of the agreement, the specific types of confidential information, the breach penalties, the applicability to parties and successors, the governing law, and the provisions regarding non-competition, amendment, and termination. Additionally, it incorporates the additional information about the agreement’s validity period and the option for legal recourse in case of non-compliance. Therefore, the merged NDA <S> scores a 7 for redundancy and a 10 for retained information.

Final Overall Score (Harmonic Mean of Averages):

7.78

Table 49: Merging 4 example NDAs: **Final Merged NDA**

NON-DISCLOSURE AGREEMENT (NDA)

1. Agreement between [Your Company Name] and [Recipient Name] on [Date] for the purpose of [specific project or purpose].
2. ”Confidential Information” includes all potentially commercially valuable information, such as software development tactics, processes, in-house research results, system designs, architectural schematics, proprietary computational algorithms, internal software blueprints, and intellectual property awaiting patents.
3. Receiving party is obligated to protect the Confidential Information, use it solely for the disclosed purpose, and not disclose it without consent.
4. Breach penalties include injunctive relief, other remedies, and a fine per breach (e.g., $200,000 or $280,000 for each infringement, depending on the agreement).
5. The Agreement applies to the Parties and their successors and assigns. It contains all related agreements and lack of enforcement doesn’t imply waiver.
6. The Agreement is under the laws of [State] and is valid for a specified period (e.g., two years) unless terminated earlier with written notice (e.g., 30 days).
7. The Receiving Party agrees not to engage in any competing business against the Disclosing Party during the agreement and a specified period after its termination (e.g., one year).
8. The Agreement can only be amended if written and signed by both parties, and overrules previous related agreements.
9. The Agreement is legally enforceable and binding for signatories and successors.
10. The Agreement is governed by the laws of [State].

Signed as of the Effective Date by [Your Company Name] - Disclosing Party [Recipient Name] - Receiving Party, along with the other parties involved.

F Evaluation - GoT Configurations

We detail the concrete operations that GoT was configured with to solve the set intersection and sorting use cases.

Listing 1: GoT configuration for the set intersection use case with 32 elements

```
Generate(k=1) # Split second set into two halves of 16 elements
foreach subset:
  Generate(k=5) # Determine intersected subset of subset and first input set
  Score(k=1) # Score locally the intersected subsets
  KeepBestN(1) # Keep the best intersected subset
Aggregate(10) # Merge both intersected subsets
Score(k=1) # Score locally the intersected result sets
KeepBestN(1) # Keep the best result
GroundTruth() # Compare to precomputed result
```

Listing 2: GoT configuration for the set intersection use case with 64 elements

```
Generate(k=1) # Split second set into four parts of 16 elements
foreach subset:
  Generate(k=5) # Determine intersected subset of subset and first input set
  Score(k=1) # Score locally the intersected subsets
  KeepBestN(1) # Keep the best intersected subset
merge step 1:
  Aggregate(10) # Merge intersected subsets 1 and 2
  Score(k=1) # Score locally the intersected result sets
  KeepBestN(1) # Keep the best result
merge step 2:
  Aggregate(10) # Merge intersected subsets 3 and 4
  Score(k=1) # Score locally the intersected result sets
  KeepBestN(1) # Keep the best result
final merge:
  Aggregate(10) # Merge intermediate intersected subsets from merge step 1 and 2
  Score(k=1) # Score locally the intersected result sets
  KeepBestN(1) # Keep the best result
GroundTruth() # Compare to precomputed result
```

Listing 3: GoT configuration for the set intersection use case with 128 elements

```
Generate(k=1) # Split second set into eight parts of 16 elements
foreach subset:
  Generate(k=5) # Determine intersected subset of subset and first input set
  Score(k=1) # Score locally the intersected subsets
  KeepBestN(1) # Keep the best intersected subset
merge step 1:
  Aggregate(5) # Merge intersected subsets 1 and 2
  Score(k=1) # Score locally the intersected result sets
  KeepBestN(1) # Keep the best result
merge step 2:
  Aggregate(5) # Merge intersected subsets 3 and 4
  Score(k=1) # Score locally the intersected result sets
  KeepBestN(1) # Keep the best result
merge step 3:
  Aggregate(5) # Merge intersected subsets 5 and 6
  Score(k=1) # Score locally the intersected result sets
  KeepBestN(1) # Keep the best result
merge step 4:
  Aggregate(5) # Merge intersected subsets 7 and 8
  Score(k=1) # Score locally the intersected result sets
```

Listing 4: GoT configuration for the set intersection use case with 128 elements (cont.)

```
  KeepBestN(1) # Keep the best result
merge step 5:
  Aggregate(5) # Merge intermediate intersected subsets from merge step 1 and 2
  Score(k=1) # Score locally the intersected result sets
  KeepBestN(1) # Keep the best result
merge step 6:
  Aggregate(5) # Merge intermediate intersected subsets from merge step 3 and 4
  Score(k=1) # Score locally the intersected result sets
  KeepBestN(1) # Keep the best result
final merge:
  Aggregate(5) # Merge intermediate intersected subsets from merge step 5 and 6
  Score(k=1) # Score locally the intersected result sets
  KeepBestN(1) # Keep the best result
GroundTruth() # Compare to precomputed result
```

Listing 5: GoT configuration for the sorting use case with 32 elements

```
Generate(k=1) # Split list into two halves of 16 elements
foreach list part:
  Generate(k=5) # Sort list part
  Score(k=1): # Score partially sorted list
  KeepBestN(1): # Keep the best partially sorted list
Aggregate(10) # Merge both partially sorted lists
Score(k=1) # Score locally the sorted result lists
KeepBestN(1) # Keep the best result
Generate(k=10) # Try to improve solution
Score(k=1) # Score locally the sorted result lists
KeepBestN(1) # Keep the best result
GroundTruth() # Compare to precomputed result
```

Listing 6: GoT configuration for the sorting use case with 64 elements

```
Generate(k=1) # Split list into four parts of 16 elements
foreach list part:
  Generate(k=5) # Sort list part
  Score(k=1) # Score partially sorted list
  KeepBestN(1) # Keep the best partially sorted list
merge step 1:
  Aggregate(10) # Merge partially sorted lists 1 and 2
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1) # Keep the best result
  Generate(k=5) # Try to improve the partial solution
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1) # Keep the best result
merge step 2:
  Aggregate(10) # Merge partially sorted lists 3 and 4
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1) # Keep the best result
  Generate(k=5) # Try to improve the partial solution
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1) # Keep the best result
final merge:
  Aggregate(10) # Merge partially sorted lists from merge step 1 and 2
  Score(k=1) # Score locally the sorted result lists
  KeepBestN(1) # Keep the best result
  Generate(k=10) # Try to improve solution
  Score(k=1) # Score locally the sorted result lists
  KeepBestN(1) # Keep the best result
GroundTruth() # Compare to precomputed result
```

Listing 7: GoT configuration for the sorting use case with 128 elements

```
Generate(k=1) # Split list into eight parts of 16 elements
foreach list part:
  Generate(k=5) # Sort list part
  Score(k=1) # Score partially sorted list
  KeepBestN(1) # Keep the best partially sorted list
merge step 1:
  Aggregate(10) # Merge partially sorted lists 1 and 2
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1) # Keep the best result
  Generate(k=5) # Try to improve the partial solution
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1) # Keep the best result
merge step 2:
  Aggregate(10) # Merge partially sorted lists 3 and 4
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1) # Keep the best result
  Generate(k=5) # Try to improve the partial solution
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1) # Keep the best result
merge step 3:
  Aggregate(10) # Merge partially sorted lists 5 and 6
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1) # Keep the best result
  Generate(k=5) # Try to improve the partial solution
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1) # Keep the best result
merge step 4:
  Aggregate(10) # Merge partially sorted lists 7 and 8
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1) # Keep the best result
  Generate(k=5) # Try to improve the partial solution
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1) # Keep the best result
merge step 5:
  Aggregate(10) # Merge partially sorted lists from merge step 1 and 2
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1) # Keep the best result
  Generate(k=5) # Try to improve the partial solution
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1) # Keep the best result
merge step 6:
  Aggregate(10) # Merge partially sorted lists from merge step 3 and 4
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1) # Keep the best result
  Generate(k=5) # Try to improve the partial solution
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1 # Keep the best result
final merge:
  Aggregate(10) # Merge partially sorted lists from merge step 5 and 6
  Score(k=1) # Score locally the partially sorted result lists
  KeepBestN(1) # Keep the best result
  Generate(k=10) # Try to improve solution
  Score(k=1) # Score locally the sorted result lists
  KeepBestN(1) # Keep the best result
GroundTruth() # Compare to precomputed result
```