Demystifying Chains, Trees, and Graphs of Thoughts

Maciej Besta[1†], Florim Memedi[1], Zhenyu Zhang[1], Robert Gerstenberger[1],
Guangyuan Piao[2], Nils Blach[1], Piotr Nyczyk[3], Marcin Copik[1],
Grzegorz Kwaśniewski[1], Jürgen Müller[4], Lukas Gianinazzi[1], Ales Kubicek[1],
Hubert Niewiadomski[3], Aidan O'Mahony[2], Onur Mutlu[1], Torsten Hoefler[1]

[1]ETH Zurich   [2] Dell   [3]Cledar   [4]BASF SE

Abstract—The field of natural language processing (NLP) has witnessed significant progress in recent years, with a notable focus on improving large language models' (LLM) performance through innovative prompting techniques. Among these, prompt engineering coupled with *structures* has emerged as a promising paradigm, with designs such as Chain-of-Thought, Tree of Thoughts, or Graph of Thoughts, in which the overall LLM reasoning is guided by a structure such as a graph. As illustrated with numerous examples, this paradigm significantly enhances the LLM's capability to solve numerous tasks, ranging from logical or mathematical reasoning to planning or creative writing. To facilitate the understanding of this growing field and pave the way for future developments, we devise a general blueprint for effective and efficient LLM reasoning schemes. For this, we conduct an in-depth analysis of the prompt execution pipeline, clarifying and clearly defining different concepts. We then build the first taxonomy of *structure-enhanced* LLM reasoning schemes. We focus on identifying fundamental classes of harnessed structures, and we analyze the representations of these structures, algorithms executed with these structures, and many others. We refer to these structures as *reasoning topologies*, because their representation becomes to a degree spatial, as they are contained within the LLM context. Our study compares existing prompting schemes using the proposed taxonomy, discussing how certain design choices lead to different patterns in performance and cost. We also outline theoretical underpinnings, relationships between prompting and other parts of the LLM ecosystem such as knowledge bases, and the associated research challenges. Our work will help to advance future prompt engineering techniques.

Index Terms—Prompt Engineering, Prompting, Prompting Topology, Chain-of-Thought, Tree of Thoughts, Graph of Thoughts, Large Language Models, LLMs, Vision-Language Models, VLMs, Generative AI, Chain Prompting, Tree Prompting, Graph Prompting.

✦

1 INTRODUCTION

Large Language Models (LLMs) have become a dominant tool in modern machine learning (ML). Originating from simple Natural Language Processing (NLP) tasks [34], [83], [157], their far-extending potential has been quickly applied in other fields, such as logical reasoning [51], planning [193], medicine [182], and many others. Since the primary communication medium with LLMs is natural language, prompt engineering has become a new area of study that gained widespread attention and importance [156], [201]. First, it democratizes the access to LLMs and to the overall generative AI landscape, by being easy to use and try by anybody. Second, it is cost-effective and does not require fine-tuning or pre-training, which are expensive and time-consuming.

Crafting LLM queries to increase both the accuracy of outcomes as well as cost-effectiveness in tasks such as logical or algebraic queries is challenging. Despite continuous advances in the size and cognitive power of LLMs, solving elaborate tasks with a single straightforward prompt yields imprecise or plain incorrect results due to the left-to-right, one-token-at-a-time nature of generative Transformer models [195]. Therefore, recent works focus on guiding LLMs towards the final solution through intermediate steps. Examples of such schemes include Chain-of-Thought (CoT) [195], Tree of Thoughts (ToT) [213], Graph of Thoughts (GoT) [10], AutoGPT [160], ReAct [214], or LLMCompiler [110]. This line of works increases the performance of the LLM reasoning.

Yet, despite all these advancements, state-of-the-art schemes still exhibit numerous limitations. First, they are still limited to simple tasks such as Game of 24 – it is critical to further enhance prompting to enable solving complex multifaceted tasks. Moreover, state-of-the-art prompting schemes often entail high inference costs [10], [213]. Third, designing, developing, maintaining, and extending these schemes is hard. On one hand, this is due to the rapid development and enrichment of the "LLM ecosystem" that must be seamlessly integrated into the prompting pipeline. This includes retrieval-augmented generation (RAG), accessing the Internet, executing Python scripts, fine-tuning, and others. On other hand, different concepts related to the LLM reasoning are not well-defined, hindering effective design of new more powerful schemes. For example, while many schemes rely on the notion of the LLM *thought*, it is not clear how it relates to concepts such as a prompt.

To address the above issues, we first identify and crystallize fundamental building blocks and concepts in the general prompt execution pipeline. Then, we analyze and clarify these blocks and concepts in the context of recent schemes such as CoT, ToT, and GoT (**contribution #1**). Our study is based on a broad analysis of recent works on LLM reasoning. Then, we use the gained insights to develop a general blueprint and a taxonomy of the LLM reasoning schemes, focusing on how the underlying *structure of reasoning* can be used to facilitate more efficient, effective, and
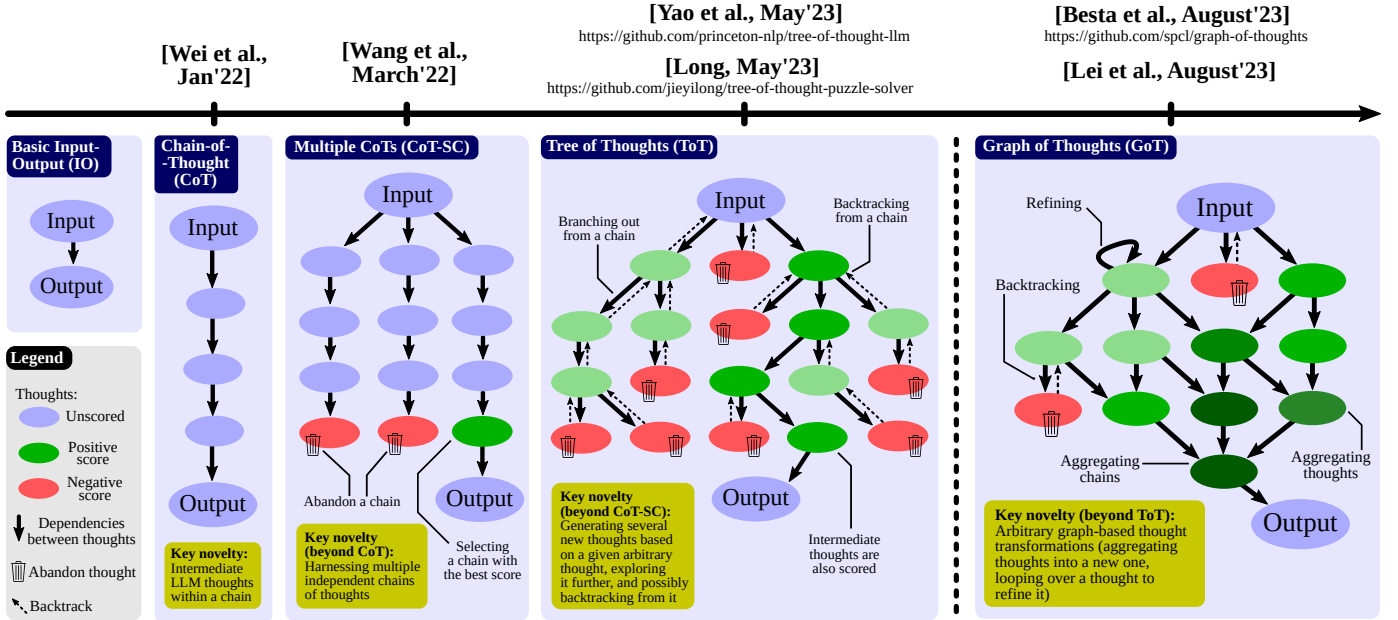
---
†*Corresponding author*

Fig. 1: **Evolution of reasoning topologies used in prompting schemes**.

productive prompting (**contribution #2**). For this, we observe that the reasoning process in many recent prompting schemes can be modeled as a *graph*. While the nature of interacting with the LLM is temporal, the representation of the graph structure behind the LLM reasoning is periodically merged with the LLM context, becoming – to a degree – spatial, thus forming *different topologies*. These topologies can be a plain path graph (as in CoT [195]), multiple parallel path graphs with a single root (as in CoT with Self-Consistency) [190], a tree (as in ToT [213]), or an arbitrary graph (as in GoT [10]). We then use our taxonomy to survey and analyze existing prompting schemes (**contribution #3**). We dissect these schemes into fundamental aspects such as the class of graphs (i.e., the topology) used to model the reasoning process, the representation of this reasoning, or the encoding of the reasoning schedule. We focus on investigating which classes of schemes offer more performance in terms of the accuracy of predictions, the latency of execution, or the cost effectiveness (**contribution #4**). We finally list open challenges and potential for novel research directions (**contribution #5**).

2 EVOLUTION OF REASONING TOPOLOGIES

We first summarize the evolution of reasoning topologies; see Figure 1 for an overview. For the sake of brevity, we do not yet precisely define the used terminology, instead relying on terms broadly used in the literature. In Sections 3–4, we introduce and discuss precise naming.

In the basic **Input-Output (IO)** prompting, the LLM provides a final reply immediately upon receiving the user initial prompt. There are no intermediate steps in the LLM reasoning. **Chain topologies**, introduced in Chain-of-Thought by Wei et al. [195], improve upon IO prompting by incorporating explicit intermediate "steps of reasoning" in addition to the input and output. **Chain-of-Thought with Self-Consistency** (CoT-SC) [190] improves upon CoT by introducing several independent reasoning chains, originating

from the same initial input. Then, the best outcome from the final thoughts is chosen, according to a predefined function S. The driving idea is to harness the randomness within the LLM reasoning, as it can generate different thoughts from the same prompt.

Tree of Thoughts (ToT) [133], [213] elevates the CoT limitations by allowing prompt branching at any point of the chain of thoughts. Therefore, different exploration paths are not fundamentally independent, like in CoT-SC, but a chain of thoughts can branch during the reasoning process to explore different options. 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 (e.g., BFS or DFS).

Finally, **Graph of Thoughts (GoT)** [10] enables arbitrary reasoning dependencies between generated thoughts. Similarly to ToT, every thought can generate multiple child thoughts. However, each thought can also have *multiple* parents, which can form an aggregation operation. GoT, allowing both branching (thoughts with out-degree > 1) and aggregation (thoughts with in-degree > 1) operations, can express – for example – reasoning patterns resembling dynamic programming, where GoT subgraphs are responsible for solving subproblems, which are then combined to form a final solution.

3 ESSENCE OF GENERAL PROMPT EXECUTION

We first summarize general prompt execution by giving a detailed overview of the prompting pipeline (Section 3.1) and then establishing a functional formulation for *any* prompting scheme (Section 3.2). This formulation will facilitate our subsequent analysis of reasoning topologies.
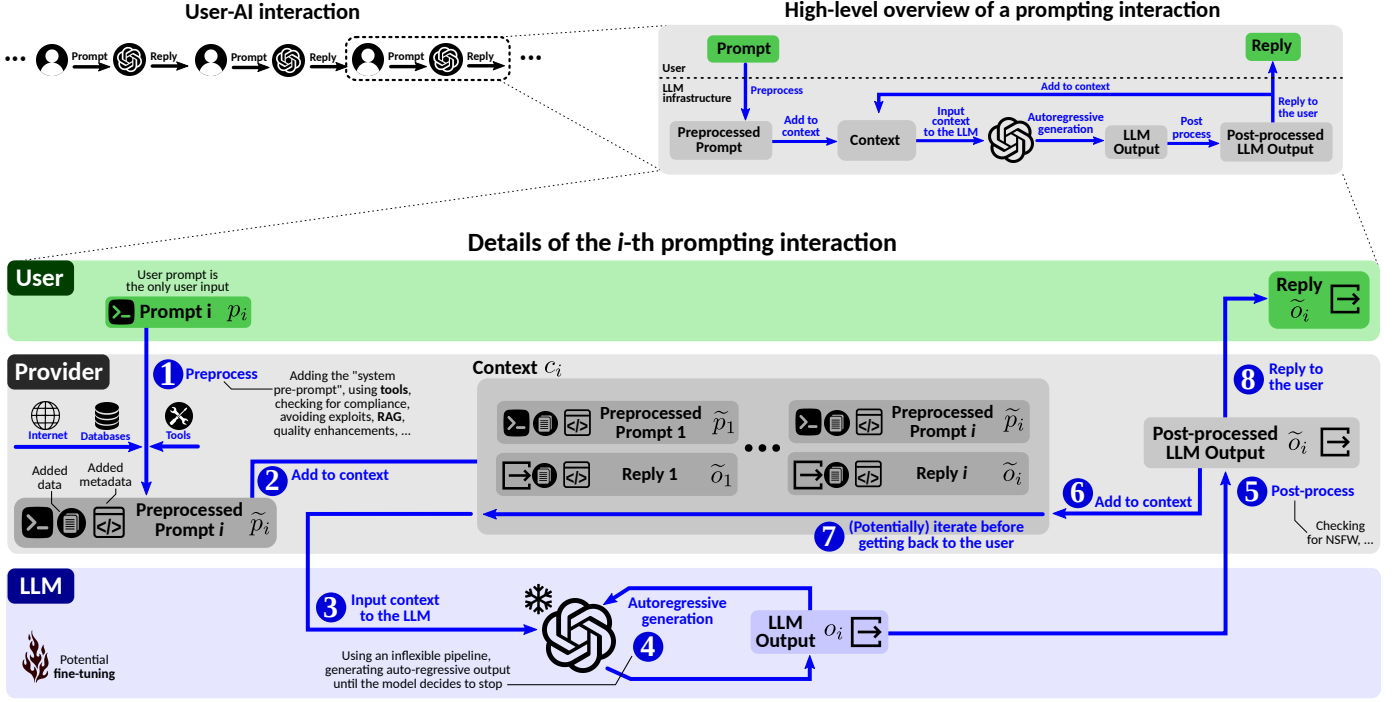
Fig. 2: **Overview of a general prompting pipeline.**

3.1 Basic Prompting Pipeline

Figure 2 summarizes the prompting pipeline. On the left side of the figure, it shows a high-level user–AI interaction, which consists of a series of exchanges of information between the user and the LLM infrastructure; the user sends *prompts* while the LLM sends back *replies*.

An individual i-th prompting interaction, pictured in the main central part of Figure 2, starts with the user sending a prompt p_i. Before being fed to the model, the prompt can be preprocessed by the LLM provider **1**, becoming $\widetilde{p}_i$. This could include adding a "system pre-prompt" or some additional metadata, checking for the compliance with some policies, enhancing the prompt quality, conducting retrieval augmentation, or including outcomes from running external tools such as Python scripts or accessing the Internet. The preprocessed prompt is added to the LLM context **2**, which then is fed to the model **3**. This results in the autoregressive output generation **4**. The model is usually frozen such that its weights do not change. In some considered prompting schemes, the model can also be fine-tuned. The output o_i is potentially post-processed **5**, which could involve running o_i through additional neural layers (e.g., for sentiment analysis) or other forms of post-processing on the provider side, such as checking for NSFW, adding more metadata, and other operations. The post-processed output $\widetilde{o}_i$ is also added to the context **6** and it is sent back to the user as a reply **8**. Note that $\widetilde{o}_i$ could also be potentially fed back to the model directly, for additional iterations, before getting back to the user **7**.

3.2 Functional Formulation & Building Blocks

We formalize the basic prompting pipeline from Section 3.1. This allows us to crystallize its fundamental building blocks, facilitating future optimizations and propelling both efficient and effective designs. The fundamental *functional* building blocks are f_{pre} (for the prompt preprocessing **1**), f_{post} (for the post-processing of the LLM output **5**), LLM (for the auto-generative LLM execution **4**), f_c (for determining how the context is updated in stage **2** and f_c' (for determining how the context is updated in stage **6**). For this, we observe that the i-th prompting interaction (for $i = 1, ...$ and $c_0' = \{\}$) can be formally described as

$$\widetilde{p}_i = f_{\text{pre}}(p_i) \tag{1}$$
$$c_i = f_c(\widetilde{p}_i, c_{i-1}') \tag{2}$$
$$o_i = \text{LLM}^X(c_i) \tag{3}$$
$$\widetilde{o}_i = f_{\text{post}}(o_i) \tag{4}$$
$$c_i' = f_c'(c_i, \widetilde{o}_i) \tag{5}$$

where

- p_i is the user prompt in the i-th prompting interaction,
- $f_{\text{pre}}(p_i)$ is a preprocessing transformation applied to p_i. It may involve **Retrieval Augmented Generation (RAG)**, executing a script, accessing the Internet, and using other **tools**,
- $\widetilde{p}_i$ is the preprocessed version of the i-th prompt,
- c_i is the context at the beginning of the i-th prompting interaction (after executing f_c),
- c_i' is the context after executing f_c' of the i-th prompting interaction (note that in the actual implementation, c_i and c_i' would be referring to the same data structure); note that $c_0' = \{\}$,
- o_i is the output of the auto-generative execution of a given LLM X (e.g., LLaMA),
- $f_{\text{post}}(o_i)$ is a post-processing transformation applied to o_i; this may involve additional neural layers (e.g., for sentiment analysis), checking for compliance with guidelines, and others. While most existing schemes do

not focus on this part, we expect that in the future, the post-processing transformation could also – similarly to preprocessing – involve executing a script, accessing the Internet, RAG, and others,

- $\widetilde{o}_i$ is the post-processed LLM output o_i; $\widetilde{o}_i = f_{\text{post}}(o_i)$,
- f_c, f'_c are transformations that determine the exact form of updating the context.

3.3 Implementing Building Blocks

The provided building blocks can serve as the basis for productive implementations of prompting baselines on different architectures. For example, scheduling different parts of the prompting pipeline in the cloud setting could be done using the granularity of these blocks: a lightweight post-processing f_{post} could execute a fast function, while a longer and stateful RAG operation within f_{pre} could be automatically placed on EC2.

The details of f_{pre}, LLM^X, f_{post}, and f_c depend on the specific LLM infrastructure. In general, they can be used to implement different parts of the generative AI ecosystem. For example, most of RAG-based frameworks would implement **RAG** in f_{pre}. Similarly, adding a **system pre-prompt** can be implemented as a part of f_{pre}. The details of how the context is updated, or how some of its parts are removed when the input length reaches its limit, are specified in f_c.

In many cases, it is the user responsibility to specify the behavior of $f_{\text{pre}}, f_{\text{post}}, f_c$; this is the case – for example – with LLaMA or when using the OpenAI API. Contrarily, when interacting with commercial services such as ChatGPT, these transformations are defined and implemented on the LLM infrastructure side.

4 ESSENCE OF REASONING TOPOLOGIES

We now crystallize different concepts in the area of reasoning topologies, and relate it to the general prompting pipeline and the functional formulation from Section 3.

4.1 What Is a Thought and a Reasoning Topology?

Many works use the term "thought". Yet, its precise meaning can differ, depending on the setting. For example, in CoT, a thought refers to a statement within a paragraph that contains a part of the reasoning process aimed at solving the input task. We show this in the top part of Figure 3. In ToT, in some tasks, such as Game of 24, a thought means an intermediate or a final solution to the initial question. However, in creative writing, it could be a plan of solving the input task, or a passage of text. In GoT, a thought contains a solution of the input task (or of its subtask). For example, it can be a subset of documents to be summarized, or a sequence of numbers to be sorted.

To encompass all these cases, we define a thought to be a *semantic unit of task resolution, i.e., a step in the process of solving a given task.* All the above examples fall into this definition: a step in task resolution can be a statement, a plan, a text passage, a set of documents, or a sequence of numbers. **We model thoughts with nodes; edges between nodes correspond to dependencies between these thoughts.** The details of these dependencies are also use case specific. For example, when generating a paragraph of text, if a given

paragraph y is a refined version of a previous version x, then x and y become nodes in the topology, and there is an edge from x to y indicating that y depends on x. If the task is to sort a sequence of numbers, and the strategy is based on splitting the sequence into sub-sequences, sorting them independently, and merging, then the initial sequence could be modeled as a node x, and the sub-sequences would form further nodes $y, z, ...$, with edges $(x, y), (x, z), ...$ from x to all the nodes modeling sub-sequences. *Now, a reasoning topology is a graph of these nodes and edges.*

Formally, a topology can be defined as $G = (V, E)$, where V is a set of nodes modeling thoughts, and E is a set of edges between these nodes, modeling reasoning dependencies between thoughts. Such a graph-theoretic approach to reason about chains, trees, and graphs of thoughts facilitates devising more efficient reasoning schemes. For example, when aiming for minimizing the latency of solving a given task, one would attempt to devise a topology with low distance between the input and output nodes.

### 4.2 Semantic Roles of Thoughts & Topologies

Graph nodes can model different aspects of reasoning. For example, in writing tasks, some nodes model plans of writing a paragraph, while other nodes model the actual paragraphs of text. We refer to such aspects as different **semantic roles**. As already observed in the prompting literature [10], semantic roles can also be modeled with graph theory, namely with heterogeneous graphs. This enables harnessing a powerful machinery for novel LLM reasoning works. For example, one could consider using some of the heterogeneous graph learning methods [139], [178], [220] in future prompting approaches.

4.3 Fundamental Use Cases of Thoughts & Topologies

We identify two fundamental use cases of thoughts and topologies: in-context examples and reasoning steps that bring us towards a solution. In a topology of thoughts, a node v is reachable from another node u, if there exists a path from u to v. If a node is reachable from the node modeling the input task statement, we call such a node a **solution node**, and the corresponding topology is a **solution topology**. However, certain nodes may be not reachable from the input node. For example, a user may provide (in their prompt) in-context examples that form a small topology, which are not a step in the reasoning towards solving the input task, but merely examples. We will refer to such thoughts and topologies as thoughts and topologies of **in-context examples**. Examples of the two use cases can be found in Figure 3. Solution thoughts and topologies are **marked with the blue color** while in-context examples are **marked with the green color**. Topologies of in-context examples do not span beyond a single prompt. Solution topologies, on the other hand, can span across many prompts and replies.

Distinguishing between solution and in-context example thoughts and topologies can enable more effective and efficient LLM reasoning schemes. For example, a graph topology has to be represented in a certain way. Now, knowing that topologies of in-context examples are usually limited to a single prompt, while solution topologies usually stretch
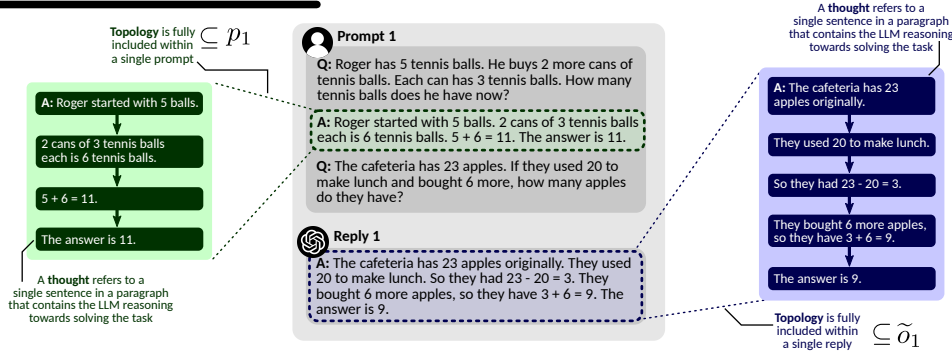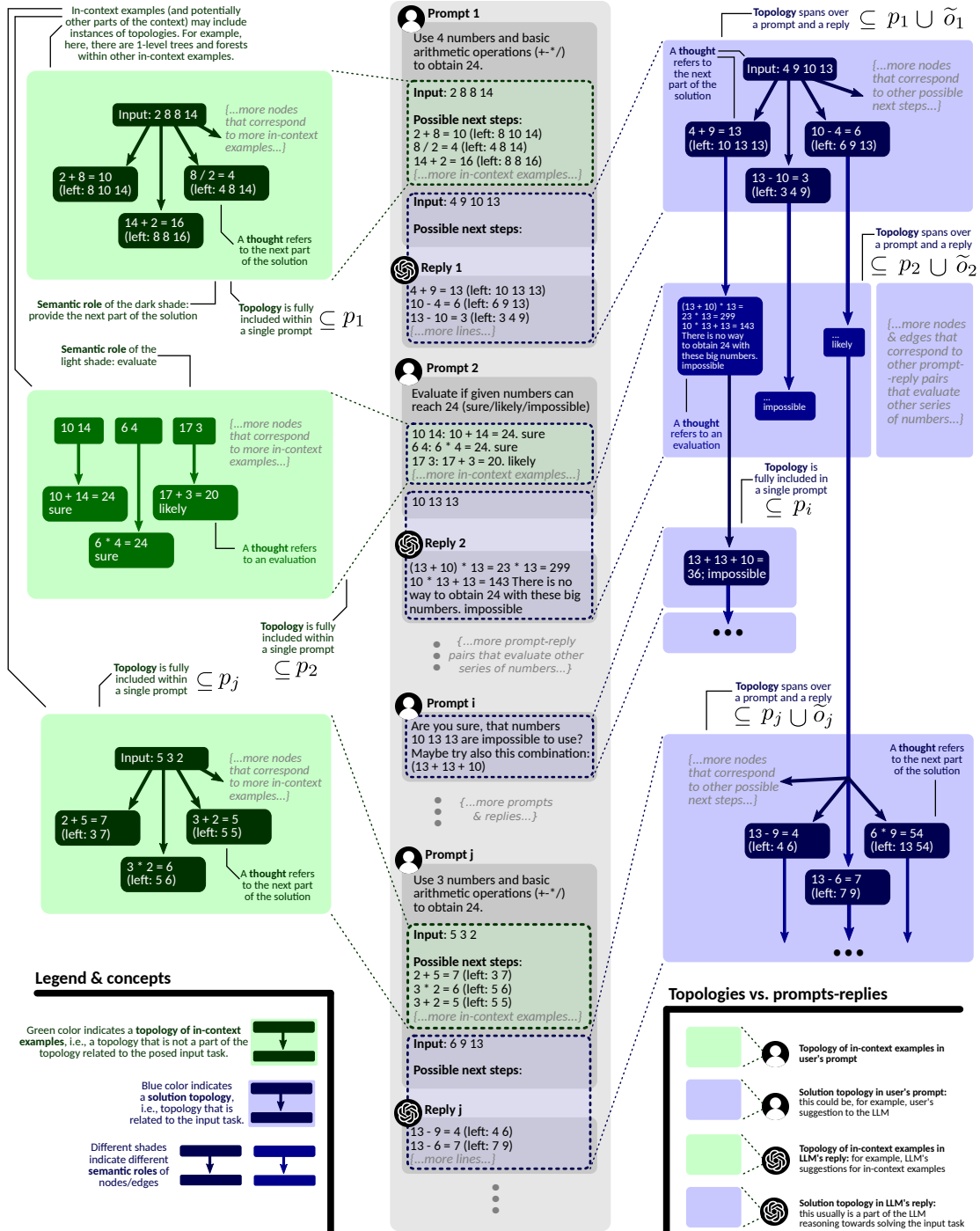
Fig. 3: **Overview and examples of how reasoning topologies correspond to the user–AI prompting interactions.**

beyond an individual prompt or LLM reply, one could use different representations for these two topology classes, in order to minimize token utilization in each of them.

Both topologies can collectively be modeled also as a graph, with multiple *components* corresponding to topologies that are not connected with one another. To further facilitate future optimizations, one could potentially harness a *hypervertex* model [31], in which arbitrary subgraphs can be modeled as individual nodes called hypervertices. In such a view, one could model each separate topology of in-context examples as a hypervertex, connected with other nodes or hypervertices with hyperedges. This approach could be harnessed to provide a theoretical framework for optimizing the holistic performance of a reasoning scheme, including its ingredients such as in-context examples, in relation to all other scheme ingredients.

4.4 Functional Formulation of Reasoning Topologies

The LLM reasoning that harnesses topologies is formulated in the exactly same way as described in Section 3. However, one has to take into account the fact that prompts p_i, replies $\widetilde{o}_i$, and context c_i in general, they all contain thoughts and their dependencies. Now, the exact way in which a topology is mapped to p_i, $\widetilde{o}_i$, and c_i, depends on a specific prompting scheme. For example, in the i-th prompting interaction of CoT, a reasoning topology T is a subset (potentially a proper subset) of either the LLM reply $\widetilde{o}_i$ (when T is a solution topology), or of the user prompt p_i, as an in-context example (when T is a topology of in-context examples). In many tree and graph based schemes, however, this mapping is not so simple, and a topology can span across multiple prompts and replies. We illustrate these examples in Figure 3.

When developing an LLM reasoning scheme that harnesses reasoning topologies, one needs to specify the details of such mappings, but also build the representation of that topology, the schedule of traversing the topology, and many others. To facilitate devising future LLM reasoning schemes, we now provide a blueprint that clearly defines all these aspects and how they can be instantiated.

4.5 A Blueprint for LLM Reasoning

We identify the following fundamental aspects of an LLM reasoning scheme that harnesses a topology: ❶ *topology class* (the structure of connections between intermediate steps of the LLM reasoning, Section 4.5.1), ❷ *topology scope* (the mapping between the topology and prompts/replies/context, Section 4.5.1), ❸ *topology representation* (how a given topology is represented within a prompt/reply/context, Section 4.5.1), ❹ *topology derivation* (how a given topology is obtained, Section 4.5.1), ❺ *reasoning schedule* (how is a given topology traversed to conduct the LLM reasoning, Section 4.5.2), ❻ *schedule representation* (how is a given schedule represented within a prompt/thought, Section 4.5.2), and ❼ *harnessed parts of the AI pipeline* (what parts of the generative AI pipeline, beyond prompting, are used, Section 4.5.3). We picture the blueprint in Figure 4, and we analyze existing schemes with respect to this blueprint[1], in Table 1.

[1]We encourage participation in this analysis. In case the reader is in possession of additional information relevant for the analyzed schemes, the authors would welcome the input.

The provided blueprint and taxonomy are applicable to both solution and in-context example topologies. For example, a topology modeling an individual in-context example can have its own representation, schedule, etc.. However, for clarity, we will focus on applying the blueprint and taxonomy mostly to solution topologies.

4.5.1 Topology of Reasoning

A reasoning scheme can harness different **topologies** for the LLM reasoning process. Here, we distinguish **chains**, **trees**, and **graphs**. Note that both a chain and a tree are each a special case of a connected graph: a tree is an acyclic connected undirected graph, and a chain is a path graph (i.e., a tree where each node has at most one child); see Part ❶ of Figure 4. Still, we treat them separately because they differ in their effectiveness for different prompting tasks [10], [133], [213]. Under this view, the plain IO prompt can be viewed as a single node graph.

Second, we observe that these topologies can be harnessed *within an individual prompt or a reply* (**single-prompt topology**), but also *across prompts or replies* (**multi-prompt topology**); see Part ❷ of Figure 4.

An important aspect is the **representation** of the topology, see Part ❸ of Figure 4. The representation can be **implicit** (the nodes and edges are not specified explicitly) or **explicit** (nodes and edges are stated explicitly). Explicit representations vary and include a set of triples [17] or a description of nodes and edges in natural text. Implicit representation depends on a scheme – for example, it could be a textual recipe that prescribes generating the next reasoning steps. Finally, we also identify **how the topology is derived** – for example, it can be constructed by the user or by the LLM itself (Part ❹ of Figure 4). Specifically, derivation of a multi-prompt topology can be manual (fixed by user before the LLM reasoning), automatic (decided dynamically by the LLM), or semi-automatic (the overall reasoning structure is predefined before the LLM reasoning starts, but the user/LLM have some control over the structure as well during the actual reasoning).

4.5.2 Reasoning Schedule

The reasoning topology forms the "skeleton" for the LLM reasoning, effectively prescribing the algorithm for solving a given task. However, for a given fixed topology, many prompting schemes offer different approaches for the execution of the intermediate reasoning steps. For example, ToT harnesses Breadth-First Search (BFS) or Depth-First Search (DFS). This motivates us to introduce another dimension of structure-enhanced reasoning, namely the *reasoning schedule*. This schedule prescribes how the topology of reasoning is going to be processed (Part ❺ of Figure 4).

Whenever the schedule is specified, it can be **represented** in different ways. It can be a description in a natural language, a code specification, in-context examples, or others (Part ❻ of Figure 4). Finally, as with the topology, the schedule itself could also be determined using different methods, for example by the LLM on-the-fly, or pre-determined (e.g., as the fixed BFS schedule).

Fig. 4: **Taxonomy and blueprint for structure-enhanced reasoning.**

4.5.3 Beyond Prompting

Many schemes go beyond pure prompting LLMs. This may include **pre-training**, **fine-tuning**, **retrieval**, **tools**, or different **modalities** (Part ❼ of Figure 4). We also consider this aspect, as it provides insights into the integration of reasoning topologies with other mechanisms in the AI pipeline beyond plain prompting interactions.

5 REASONING WITH CHAINS

We now proceed to investigate in more detail individual schemes that use chain topologies. We analyze these works with respect to our blueprint and taxonomy in the top part of Table 1 (detailed descriptions of each individual scheme are provided in the appendix) We also illustrate fundamental concepts introduced in these works, namely *multi-step reasoning, zero-shot reasoning, planning & task decomposition, task preprocessing, iterative refinement, and tool*

utilization. We finish this section with a comparative analysis and illustrations of example topology representations.

5.1 Multi-Step Reasoning

The concept of multi-step reasoning was first introduced through the seminal **Chain-of-Thought (CoT)** [195], a single-prompt scheme, which uses topologies of in-context examples, also known as few-shot examples, to guide the LLM into reasoning step-by-step before providing the final answer. Different following works augment or adapt in-context examples to elicit different forms of reasoning steps, while still relying on the single-prompt chain topology. For instance, instead of only providing a step-by-step reasoning chain in the examples, **SelfAsk** [152] expands each step in the chain to also pose a follow-up question that is then answered in the subsequent step. Similarly, **Program of Thoughts (PoT)** [41] uses code examples, rather than the

Scheme	Topology single-prompt			Topology multi-prompt			Reasoning Schedule			AI Pipeline					Remarks
	Class	Rp.	Dv.	Class	Rp.	Dv.	Scheme	Rp.	Dv.	P	F	R	T	Modalities	
Chain-of-Thought (CoT) [195]	chain	I (text)	SA	-	-	-	-	-	-	✗	✗	✗	✗	text	
Zero-shot-CoT [112]	chain	I (text)	SA	-	-	-	-	-	-	✗	✗	✗	✗	text	
SelfAsk [152]	chain	I (text)	SA	-	-	-	-	-	-	✗	✗	◧	✗	text	
Plan-and-Solve Prompting [188]	chain	I (text)	SA	-	-	-	-	-	-	✗	✗	✗	✗	text	
Program of Thoughts (PoT) [41]	chain	I (text,code)	SA	-	-	-	-	-	-	✗	✗	✗	■	text,code,table	
Selection-Inference (SI) [51]	-	-	-	chain	E	M	linear	I	M	✗	◧	✗	✗	text	
Chain-of-Symbol (CoS) [89]	chain	I (text)	SA	chain	E	SA	linear	I	M	✗	✗	✗	✗	text	
Least-to-Most Prompting [233]	-	-	-	chain	E	SA	linear	I	M	✗	✗	✗	✗	text	
Decomposed Prompting [105]	-	-	-	chain	E	SA	linear	I	M	✗	✗	◧	◧	text	
LogiCoT [231]	chain	I (text)	SA	**tree**	E	SA	linear	I	M	✗	✗	✗	✗	text	
SELF-REFINE [140]	-	-	-	chain	E	SA	linear	I	M	✗	✗	✗	✗	text	
Reflexion [168]	-	-	-	chain	E	SA	linear	I	M	✗	✗	✗	✗	text	
Reasoning Graph Verifier (RGV) [35]	chain	I (text)	SA	**graph**	E	SA	linear	I	M	✗	✗	✗	✗	text	
Plan, Verify and Switch (PVS) [131]	chain	I (text,code)	SA	chain	E	SA	linear	I	M	✗	✗	✗	■	text,code	
Chameleon [136]	-	-	-	chain	E	SA	linear	I	M	✗	✗	■	■	text,code	
ChatCoT [45]	chain	I (text)	SA	chain	E	SA	linear	I	M	✗	✗	■	■	text	
Tree-of-Thought (ToT) [133]	tree	I (text)	M	**tree**	E	SA	**arbitrary**	E	M	✗	✗	✗	✗	text	
Tree of Thoughts (ToT) [213]	tree	I (text)	M	**tree**	E	SA	**arbitrary**	E	M	✗	✗	✗	✗	text	
Thought Decomposition [205]	tree	I (text)	M	**tree**	E	SA	**beam**†	E	SA	✗	✗	✗	◧	text,code	†stochastic
Self-Consistency with CoT [190]	chain	I (text)	M	tree (■)†	E	SA	-	-	-	✗	✗	✗	✗	text	†depth one
Creswell and Shanahan [50]	tree	I (text)	M	**tree**	E	SA	**beam**	E	A	✗	■	✗	✗	text	
Dynamic Least-to-Most Prompting [58]	tree	I (text)	M	**tree**	E	A	**bottom up**	E	A	✗	✗	◧	✗	text,code	
Algorithm of Thoughts (AoT) [166]	**tree**	I (text)	M	-	-	-	**DFS, (BFS)**	I	M	✗	✗	✗	✗	text	
Tree of Uncertain Thought (TouT) [145]	tree	I (text)	M	**tree**	E	SA	**BFS, DFS**	E	A	✗	✗	✗	✗	text	
Tree-of-Mixed-Thought [91]	tree	I (text)	M	**tree**	E	SA	**DFS**	E	A	✗	✗	✗	■	scene graphs	
Tree of Clarifications (ToC) [106]	tree (■)†	I (text)	M	**tree**	E	SA	**BFS**	E	A	✗	✗	■	✗	text	†depth one
Tree Prompting [170]	-	-	-	**tree**	E	A	**top-down**	E	A	✗	✗	✗	✗	text	
Skeleton-of-Thought (SoT) [148]	tree (■)†	I (text)	M	tree (■)†	E	A	**parallel**	E	A	✗	✗	✗	✗	text	†depth one
Branch-Solve-Merge (BSM) [162]	tree (depth one)	I (text)	M	**graph (■)†**	E	SA	**BFS**	E	M	✗	✗	✗	✗	text	†double tree (■)
Thought Propagation (TP) [218]	graph (■)	arbitrary	M	**graph (■)†**	E	SA	**BFS**	E	M	✗	✗	✗	✗	text	†double tree (■)
Socratic Questioning [154]	tree (depth one)	I (text)	M	**graph (■)†**	E	SA	**DFS**	E	M	✗	✗	✗	✗	multi	†double tree
Graph of Thoughts (GoT) [10]	graph (■)	arbitrary	M	**graph**	E	M	**arbitrary**	E	M	✗	✗	✗	✗	text	
Graph of Thought (GoT) [119]	❓	❓	❓	**graph**	E	(S)A	**DFS**	E	❓	✗	✗	✗	✗	text	
Graph-of-Thought (GoT) [215]	**graph**	I (text)	M	chain	E	M	linear	E	M	✗	■	✗	✗	text,image	
ControlLLM [132]	graph	E (json)	M	**graph**	E	M	**DFS**	E	M	✗	✗†	✗	■	text,image,video,audio	†can be used
Cumulative Reasoning [224]	graph (DAG)	I (text)	M	**graph (DAG)**	E	SA	**arbitrary**	E	M	✗	✗	✗	✗	text	
Everything of Thoughts (XoT) [57]	**graph**	❓	L	chain	E	M	linear	E	M	■	✗	✗	✗	text	
ResPrompt [99]	**graph**	I (text)	M	-	-	-	-	-	-	✗	✗	✗	✗	text	
Hypergraph-of-Thought (HoT) [212]	**hypergraph**	❓	M	-	-	-	-	-	-	✗	■	✗	✗	text,image	
BatchPrompt [124]	batch	E (text)	M	chain	E	M	linear	E	M	✗	✗	✗	✗	text	
Memory Injections [163]	-	-	-	-	-	-	-	-	-	✗	✗	✗	✗	text	

TABLE 1: **Comparison of LLM reasoning schemes with respect to the provided taxonomy (Section 4.5 and Figure 4).** "**Topology**": Details of the harnessed structure. "**Single- / Multi-prompt**": Does a given scheme support *single- / multi-prompt* topology? If yes, what is the supported **Class**, **Representation**, and **Derivation**? "**Reasoning Schedule**": Details of the harnessed **reasoning schedule**, including its specific **Scheme**, **Representation**, and **Derivation**. "**AI pipeline**": Does a given scheme harness parts of the AI pipeline beyond prompting? If yes, which ones? ("**P**": pre-training, "**F**": fine-tuning, "**R**": retrieval, '**T**": tools, "**Modalities**": modalities). When describing representations, we use the following abbreviations: "**E**": explicit, "**I**": implicit. When describing derivation, we use the following abbreviations: "**A**": automatic, "**L**": learned, "**M**": manual, "**SA**": semi-automatic. "■": full support (i.e., YES), "◧": partially [supported], "✗": no support (i.e., NO).

natural-language-based examples as in CoT, to obtain a step-by-step generated, functional Python program that can be executed to get the final result.

5.2 Zero-Shot Reasoning Instructions

Zero-shot reasoning instructions aim to elicit the same multi-step reasoning chains, but without the use of hand-tuned, problem-specific in-context examples, i.e., they eliminate chain topologies forming in-context examples. **Zero-shot-CoT** [112], an extension to CoT, achieves this by simply prompting the LLM with one sentence, *"Let's think step by step"*, or using other similar statements. Along the same lines, **PoT** can also leverage zero-shot reasoning instruction, e.g., *"Let's write a Python program step by step and return the result. Firstly we need to define the variables."*.

5.3 Planning & Task Decomposition

Both planning and task decomposition aim to break down a task into a number of manageable sub-tasks that help reaching the final solution. **Plan-and-Solve (PS) Prompting** [188], one of the key single-prompt schemes building on this concept, first divides the complex task into a chain of sub-tasks and then executes these step-by-step for the final solution. PS operates in a zero-shot, multi-step manner, thus also relies on the previous two concepts as well.

Planning and decomposition is also frequently used in multi-prompt chains. The introduction of a node for specifying the details of the decomposition at the start of a multi-prompt reasoning chain, usually not only determines the chain's depth, but also facilitates more effective reasoning approaches in the subsequent sub-steps. This allows for a finer-grained resolution of sub-tasks, enriching the overall reasoning process. Here, **Least-to-Most Prompting** [233] grows a reasoning chain where decomposition of complex tasks or questions is conducted in the first node and the sub-tasks/sub-questions are solved in the subsequent nodes. Specifically, the multi-prompt chain operates by first decomposing the original question into a list of sub-questions, which are each solved in individual sub-steps with the questions and answers of previous sub-steps included in the context. The chain terminates when the final answer is returned after answering all sub-questions in the list. Then, **Decomposed Prompting** [105] is a modular framework for a detailed decomposition of complex tasks. To generate a reasoning chain, the LLM is prompted by demonstrations comprised of sequential question-operation triplets, which form "sub-questions". In contrast to Least-to-Most prompt-

ing, this allows for the recursive breakdown of questions into simpler sub-questions until they can be solved directly, as further decomposition is a valid operation in the framework. Apart from these two above schemes, decomposition is applied in many similar works [59], [102], [217].

5.4 Task Preprocessing

The concept of task preprocessing comprises any technique that preprocesses the context of a task by updating it or rephrasing the task description itself, before taking any reasoning steps. For instance, the multi-prompt scheme **Selection-Inference (SI)** [51] is designed to tackle multi-step logical reasoning problems where all essential information is already present within the input context. The key functionality of SI lies in its recurring process of context pruning before each reasoning step. This means that it selectively filters the context to retain only the relevant information necessary for each specific subsequent step of reasoning, ensuring that the most pertinent data is always used for each decision-making stage. On the other hand, instead of pruning the context, the multi-prompt scheme **Chain-of-Symbol (CoS)** [89], specifically designed for spatial planning tasks, augments the context with condensed symbolic representations, before subsequently using those as prompts for the LLM to conduct CoT-based reasoning.

5.5 Iterative Refinement

The introduction of verification enables the reasoning frameworks to iteratively refine the generated context and intermediate results. With this strategy, the execution of chain-based reasoning is effectively extended with loops, with conditions on how many times one can loop over a node (based on the number of iterations or some terminal conditions). The concept is applied in different works [123], [126], [140], [168], [198], [231].

5.6 Tool Utilization

To better integrate multiple execution methods, more effective schemes opt to devise a plan that specifies tools for handling each sub-task, before executing the reasoning chain. Examples include **AutoGPT** [160], **Toolformer** [165], **Chameleon** [136], **ChatCot** [45], **PVS** [131] and others [230].

5.7 Analysis & Comparison of Designs

We now broadly discuss and analyze chain designs with respect to different aspects of our blueprint.

5.7.1 Topology & Its Construction

In single-prompt schemes, the entire reasoning process is executed within a single prompting round. This approach is less common for complex tasks, as it often demands sophisticated prompt engineering to encompass the entire reasoning pathway in one go. On the other hand, most chain designs employ multi-prompt schemes, where the reasoning process is segmented into multiple rounds of prompting. This allows for a more nuanced and step-wise approach to problem-solving. Key novel architectural features of chain designs include the ability to appropriately

Game of 24

User Prompt

Use numbers and basic arithmetic operations (+ - * /) to obtain 24

Q: 8 8 7 4
A: (7 - (8 / 8)) * 4 = 24

Q: 9 6 5 3
A: (5 + 3) * (9 - 6) = 24

Q: 14 8 8 2

LLM Answer

Sure, here's one way to obtain 24 using the numbers 14, 8, 8, and 2

((14 - 8) / 2) * 8 = 24

Fig. 5: GAME OF 24. **An example showing an IO scheme (effectively an implicit single-prompt single-node chain topology).** A few-shot IO prompt is used, leading to a single output directly providing the result with no intermediate reasoning steps.

decompose tasks, verify and refine intermediate solutions, preprocess initial prompts, and utilize external tools, such as Python scripts. This multifaceted approach enables LLMs to tackle more complex problems by breaking them down into smaller, more manageable components and iteratively refining the solutions.

5.7.2 Representations of Topology & Schedule

We now illustrate representative prompts that show differences between single- and multi-prompt chain-based topologies (the tasks are described in detail in the prompts of the figures). We use examples based on the well-known tasks of Game of 24, Creative Writing, and Mathematical or Logical Reasoning. For this, we illustrate an example simplest IO scheme in Figure 5 (effectively a single-node CoT) and compare it to an implicit single-prompt few-shot CoT in Figure 6, an implicit single-prompt *zero-shot* CoT in Figure 7 and to an example few-shot *explicit multi-prompt* CoT (Selection-Inference) in Figure 8.

5.7.3 Performance

We now summarize performance patterns found in chain topologies. For a more detailed performance comparison, see Appendix E.1.

Overall, in Arithmetic Reasoning, CoT significantly outperforms Input-Output (IO) prompting, with notable improvements in terms of accuracy on several datasets from benchmarks such as GSM8K (Grade School Math) [48], SVAMP (Simple Variations on Arithmetic Math word Problem) [151], and MAWPS (MAth Word ProblemS) [113], which focus on mathematical tasks. The effectiveness of CoT increases with the scale of the LLM. Variants like Zero-shot-CoT, PoT, and schemes with decomposition nodes (like Least-to-Most Prompting, PS+) show further improvements in specific datasets. Chameleon, with its table reader tools, enhances performance in tabular math questions.

In Commonsense Reasoning, CoT excels over IO prompting with a distinct advantage in datasets like StrategyQA [71]. Specialized methods like SelfAsk and the Selection-Inference framework show further improvements

Math Reasoning

User Prompt

Q: Alexis is applying for a new job and bought a new set of business clothes to wear to the interview. She went to a department store with a budget of $200 and spent $30 on a button-up shirt, $46 on suit pants, $38 on a suit coat, $11 on socks, and $18 on a belt. She also purchased a pair of shoes, but lost the receipt for them. She has $16 left from her budget. How much did Alexis pay for the shoes?

A: Let S be the amount Alexis paid for the shoes.

⬇

She spent S + 30 + 46 + 38 + 11 + 18 = S + 143.

⬇

She used all but $16 of her budget, so S + 143 = 200 - 16 = 184.

⬇

Thus, Alexis paid S = 184 - 143 = 41 for the shoes.

⬇

So the answer is 41.

Q: [more in-context examples...]

Q: Tobias is buying a new pair of shoes that costs $95. He has been saving up his money each month for the past three months. He gets a $5 allowance a month. He also mows lawns and shovels driveways. He charges $15 to mow a lawn and $7 to shovel. After buying the shoes, he has $15 in change. If he mows 4 lawns, how many driveways did he shovel?

LLM Answer

Let D be the number of driveways Tobias shoveled.

⬇

He earned $5 allowance per month for three months, so he has $5 * 3 = $15 from his allowance.

⬇

He mowed 4 lawns, each earning him $15, so he made $15 * 4 = $60 from mowing lawns.

⬇

He also earned $7 for each driveway shoveled, and he shoveled D driveways, so he made $7 * D from shoveling driveways.

⬇

In total, he has $15 (allowance) + $60 (mowing lawns) + $7D (shoveling driveways) before buying the shoes.

⬇

After buying the $95 shoes, he has $15 in change.

⬇

Therefore: $15 + $60 + $7D - $95 = $15

⋮

⬇

So, Tobias shoveled 5 driveways.

Fig. 6: MATH REASONING. **Another implicit single-prompt chain topology example, encoded with text**. It shows how few-shot CoT solves a math question from GSM8K [48]. This results in an implicit chain where individual reasoning steps can be regarded as nodes.

Creative Writing

User Prompt

Write a coherent passage of 4 short paragraphs. The end sentence of each paragraph must be:
1. It isn't difficult to do a handstand if you just stand on your hands.
2. It caught him off guard that space smelled of seared steak.
3. When she didn't like a guy who was trying to pick her up, she started using sign language.
4. Each person who knows you has a different perception of who you are.
Let's think step by step.

LLM Answer

Mastering a handstand may seem daunting at first, but the key lies in simplicity. It isn't difficult to do a handstand if you just stand on your hands. Breaking down the complex maneuver into a straightforward action can make the seemingly impossible, achievable. Sometimes, the most challenging tasks become manageable when approached with a clear and uncomplicated perspective.

⬇

In the vast expanse of outer space, surprises await even the most seasoned explorers. It caught him off guard that space smelled of seared steak. The unexpected aroma challenged preconceptions, reminding us that the universe is full of mysteries that extend beyond the visual and auditory senses. The interplay of senses in unfamiliar environments can reshape our understanding of the cosmos.

⬇

Navigating social interactions often demands creative strategies, especially when faced with unwanted advances. When she didn't like a guy who was trying to pick her up, she started using sign language. Non-verbal communication became her shield, a tool to ward off unwelcome attention without resorting to confrontation. In these moments, the power of expression transcends spoken words, allowing for a graceful escape from uncomfortable situations.

⬇

Perceptions of identity are as diverse as the individuals who form them. Each person who knows you has a different perception of who you are. It's a kaleidoscope of perspectives, shaped by unique experiences and interactions. Understanding that our multifaceted nature is reflected in the eyes of those around us can foster empathy and open the door to a richer tapestry of human connections.

Fig. 7: CREATIVE WRITING. **An example implicit single-prompt chain topology, encoded with text**. It shows the Creative Writing example from ToT [213] using zero-shot CoT prompting. This results in an implicit chain with a first node for the question and one node for each generated paragraph.

Logical Reasoning

User Prompt

If something is quiet and cold then it is smart.
All young, cold things are kind.
If something is kind then it is nice.
All smart, blue things are kind.
If something is cold then it is quiet.
If Anne is smart and Anne is kind then Anne is cold.
If something is nice and blue then it is young.
If something is nice and young then it is cold.

Anne is young.
Gary is nice.
Charlie is quiet.
Anne is blue.
[...]
Dave is smart.
Anne is quiet.
Dave is blue.
Gary is cold.
Anne is nice.

Does it imply that the statement "Dave is not quiet" is true?

LLM Answer

Selection · · · · · · · · · · · · · **Inference**

All smart, blue things are kind.
We know that Dave is smart and Dave is blue.

Dave is kind.

If something is kind then it is nice.
We know that Dave is kind.

Dave is nice.

If something is nice and blue then it is young.
We know that Dave is nice and Dave is blue.

Dave is young.

If something is nice and young then it is cold.
We know that Dave is nice and Dave is young.

Dave is cold.

If something is cold then it is quiet.
We know that Dave is cold.

Dave is quiet.

Fig. 8: LOGICAL REASONING. **An example explicit multi-prompt chain topology, encoded with text**. It demonstrates the manually derived chain topology of Selection-Inference [51]. "Selection" and "Inference" indicate independent consecutive prompts/thoughts.

In Symbolic Reasoning, CoT demonstrates near-perfect accuracy for in-context examples and substantial accuracy for out-of-domain cases for tasks like last letter concatenation and coin flip predictions. Schemes with decomposition nodes outperform CoT in more complex tasks and longer word scenarios.

Overall, across different domains, CoT and its variants show a consistent trend of outperforming basic IO prompting. The integration of additional tools, tailored prompting strategies (like few-shot or zero-shot), and the incorporation of decomposition and refinement nodes enhance the LLMs'

in multi-hop questions. Decomposition-based schemes like Least-to-Most and Decomposed Prompting achieve high accuracy in tasks requiring sequential actions or multi-faceted reasoning. ChatCoT, with its retrieval and external tool utilization, shows a 20% gain on specific datasets.

Tree Topology Variants

k-ary tree

Input

Output

Different frameworks enable using arbitrary trees, but the prevalent variant is a k-ary tree

Example designs:
Tree-of-Thought
Tree of Thoughts
[Creswell & Shanahan]
Dynamic Least-to-Most
Algorithm of Thoughts
Tree of Uncertain Thought
Tree-of-Mixed-Thought
Tree of Clarifications
Thought Decomposition

Tree of chains

Input

Output

Schemes that improved upon Chain-of-Thought, such as Chain-of-Thought with Self-Consistency, are effectively harnessing tree topology variants

Example designs:
CoT with Self-Consistency
Reasoning Graph Verifier

1-Level trees

Input

Example designs:
Skeleton-of-Thought

Binary tree

Input

Output

Example designs:
Tree Prompting

Graph Topology Variants

Directed graphs + loops

Input

Output

Example designs:
Graph of Thoughts

Branch-Merge

Input

Output

Directed graphs

Input

Output

Example designs:
Graph of Thoughts
Graph-of-Thought
ControlLLM
Cumulative Reasoning
Everything of Thoughts
ResPrompt

Hypergraphs

Input

Output

The only currently existing Hypergraph-based scheme, the Hypergraph-of-Thought, uses a limited variant of hypergraphs, in which hyperdges span only triples of vertices

Example design:
Hypergraph-of-Thought

Thought Propagation

Input

Output

Socratic Questioning

Input

Output

Fig. 9: **Variants of tree and graph prompting topologies**.

performance significantly.

6 Reasoning with Trees

We next investigate in more detail individual schemes that use tree topologies. We analyze these works with respect to our blueprint and taxonomy in the middle part of Table 1 (detailed descriptions of each individual scheme are provided in the appendix). We structure the discussion based on the harnessed topology variants, namely trees of chains, 1–level trees, and k–ary trees, see Figure 9 for details. As in chain 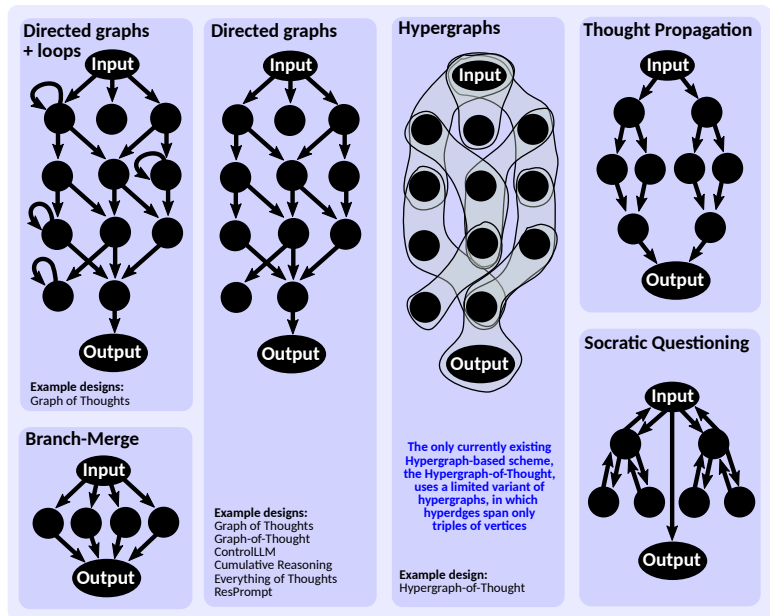schemes, we also discuss fundamental concepts introduced or harnessed in these works. Tree schemes, most importantly, introduce *exploration* (i.e., generating multiple thoughts from a given one). The purpose behind exploration is usually either *task decomposition* (which is similar to CoT, but it differs as decompositions are not limited to a single linear plan) or *sampling* (i.e., having a higher chance of obtaining a high-quality solution). Moreover, tree schemes also introduce *voting* (i.e., automatic selection of the best outcome of all the generated outputs) and they harnesses various architectural concepts also used in chain schemes, such as iterative refinement or task preprocessing. We finish this section with a comparative analysis and illustrations of example topology representations.

6.1 Trees of Chains

While trees as reasoning topologies have been explicitly established in the works by Long [133] and Yao [213], this idea has been present earlier. **Chain-of-Thought with Self-Consistency (CoT-SC)** [190] is an early scheme that harnesses the tree structure to a certain degree. Here, multiple CoTs originate from the same initial (root) prompt, forming a "tree of chains". The chain providing the best outcome to the initial question, is selected as the final answer.

6.2 Single-Level Trees

A tree-based approach has also been harnessed in **Skeleton-of-Thought (SoT)** [148], which effectively harnesses a tree with a single level of depth. This scheme aims to reduce the end-to-end generation latency of LLMs, caused by their inherent sequential decoding. Instead of generating one long continuous answer, this scheme uses a divide-and-conquer approach. In a first prompt, the LLM is instructed to generate a skeleton of the answer, i.e., a list of points that are independently answerable. Then, for each of these points, a new prompt is issued in parallel to answer just this specific part of the question. As these points are processed in parallel, the overall latency is reduced.

6.3 k–Ary Trees

Numerous schemes have harnessed more general k–ary trees. First, the **Tree-of-Thought (ToT) design by Long** [133] utilizes a tree structure to decompose a problem into subproblems and solve them using separate LLM prompts. After the LLM suggests possible next steps and corresponding partial solutions, a *checker module* decides if any of these solutions is valid, whether it can be selected as the final one, or whether it should backtrack to the previous step. All issued prompts and answers are explicitly stored as a tree structure and navigated through using a *controller module*. The LLM prompting is only used to generate the next individual steps (i.e., hops) in this tree, whereas the overall problem solving process is coordinated by the controller.

Tree of Thoughts (ToT) by Yao et al. [213] differs from the above ToT approach in using the LLM itself as a solution *evaluator* with access to all generated solutions, instead of using a programmed or learned evaluator module. This allows to rate states individually or vote across intermediate solutions to select the most promising one to continue with

the search. Both mentioned ToT approaches are a generalization of the IO, CoT, and CoT-SC prompting schemes.

Other examples in this class of topologies include **Thought Decomposition** [205] (a multi-prompt scheme based on stochastic beam search and self-evaluation), a scheme by **Creswell and Shanahan** [50] (an extension of the chain-based Selection-Inference [51]), **Dynamic Least-to-Most Prompting** [58] (an extension of least-to-most prompting with a tree-based problem decomposition and a dynamic external tree-based few-shot example selection), **Algorithm of Thoughts (AoT)** [166] (a single-prompt approach that utilizes in-context examples formulated in an algorithmic tree-based fashion), **Tree of Uncertain Thought (TouT)** [145] (an extension of ToT with local "uncertainty scores" by incorporating the variance of multiple LLM responses into the state evaluation function), **Tree-of-Mixed-Thought (TomT)** [91] (a ToT-based reasoning scheme to answer questions on visual scene graphs), or **Tree of Clarifications (ToC)** [106] (recursive prompting of an LLM to construct a tree of disambiguations for the initial question).

6.4 Analysis & Comparison of Designs

We now broadly discuss and analyze tree designs with respect to different aspects of our blueprint.

6.4.1 Topology & Its Construction

The key novel architectural feature of tree schemes is the exploration of a thought, i.e., the ability to generate multiple new steps based on a given single one. The vast majority of tree schemes are multi-prompt. Most multi-prompt schemes use a dynamic approach to building the tree topology. The details of how the topology is exactly shaped depend on the specific question. For most multi-prompt approaches, the user can adapt the tree topology to a certain degree, i.e., by varying the *branching factor* (i.e., the number of thoughts generated from a given vertex) and limiting the depth of the tree.

6.4.2 Representations of Topology & Schedule

We showcase the differences between implicit vs. explicit and single- vs. multi-prompt topologies using representative examples (the tasks are described in detail in the prompts of the figures). We continue with the driving tasks of Game of 24, Creating Writing, and Mathematical/Logical Reasoning. For this, we illustrate an implicit single-prompt tree topology elicited by AoT in Figure 10 as well as two example explicit multi-prompt tree topologies from ToT and CoT-SC in, respectively, Figure 11 and 12. Lastly, we show an example of a parallel execution schedule in Figure 13 for SoT [148].

6.4.3 Performance

We now summarize performance patterns found within tree topologies. A detailed analysis can be found in Appendix E.2.

Overall, increasing the branching factor (i.e., the number of thoughts generated from a given vertex) often leads to a higher diversity of outcomes, which can be beneficial for accuracy, but it also increases #prompts, i.e., computational cost. The most advantageous branching factor is

hard to find and it often depends on the specific problem to solve. Easily decomposable problems may benefit less from more branching than complex problems. Specifically, more complicated problems profit more from decomposing them into many/diverse sub-problems (e.g., this ensures enough diversity for self-consistency to work better). In contrast, a question that has clearly only two sub-parts does not benefit from many more subdivisions, as the additional branches then can be either redundant or wrong. Single-prompt approaches can perform better on some problems than multi-prompt approaches, while using only a single prompt compared to possibly hundreds [166].

7 REASONING WITH GRAPHS

We also analyze schemes that harness graph topologies, see the bottom part of Table 1 (detailed descriptions of each individual scheme are provided, as for chains and trees, in the appendix). Similarly to the tree analysis, we structure the discussion based on the harnessed topology variants, see Figure 9 for details. We also discuss fundamental concepts introduced or harnessed in these works. Graph schemes, most importantly, introduce *aggregation* (i.e., being able to combine multiple thoughts into a single one). The purpose behind aggregation is usually *synergy* (i.e., being able to produce an outcome better than the individual ingredients) or an effective *composition of outcomes of tasks*. Graph schemes also use architectural concepts employed in chain or tree schemes, such as exploration or iterative refinement. We conclude this section with a comparative analysis and illustrations of example representations of graph topologies.

7.1 Special Classes of Graphs

Different schemes harness certain special classes of graphs. **Branch-Solve-Merge (BSM)** [162] employs a 1–level double tree structure to first divide a problem into independently solvable sub-problems, and then combines them into a final solution. The first prompt instructs the LLM to propose sub-problems, which are then solved independently. The final prompt instructs the LLM to merge the results of the sub-problems into a single output. **Socratic Questioning** [154] is a scheme that models recursive exploration of the thought space using a tree structure. Hereby, the original question is recursively decomposed into sub-tasks until all tasks can be solved with high confidence. These results are then aggregated and propagated back up the tree to answer the original question. This results in an overall double tree reasoning topology.

7.2 Directed Graphs

Some schemes embrace a general directed graph model. **Graph of Thoughts (GoT)** [10] uses a multi-prompt approach to improve the LLM problem solving performance by decomposing a given task into sub-tasks that form a graph. This decomposition is specified as a *Graph of Operations*, which coordinates how the LLM is prompted and how the results are further used in the reasoning process. The **Graph of Thought** [119] presents a multi-prompt approach where a graph of thoughts is constructed recursively in a DFS manner by starting at the *question node* that represents

Game of 24

User Prompt

Use numbers and basic arithmetic operations (+ - * /) to obtain 24.
[in-context examples]
User:
14 8 8 2
Assistant:

LLM Answer

Trying a promising first operation:
1. 8 + 2: (14, 10, 8)
- 14 + 10: (24, 8) 32, 16, 192, 3
- 14 - 10: (8, 4) 12, 4, 32, 2
- 14 * 10: (140, 8) 148, 132, 1120, fractional
- 14 / 10: fractional
- 14 + 8: (22, 10) 32, 12, 220, fractional
- 14 - 8: (10, 6) 16, 4, 60, fractional
- 14 * 8: (112, 10) 122, 102, 1120, fractional
- 14 / 8: fractional
- 10 + 8: (18, 14) 32, 4, 252, fractional
- 10 - 8: (14, 2) 16, 12, 28, 7
- 10 * 8: (80, 14) 94, 66, 1120, fractional
- 10 / 8: fractional
Trying another promising first operation:
2. 8 / 2: (14, 8, 4)
- 14 + 8: (22, 4) 26, 18, 88, fractional
- 14 - 8: (6, 4) 10, 2, 24 = 6 * 4 -> found it!
Backtracking the solution:
Step 1:
8 / 2 = 4
Step 2:
14 - 8 = 6
Step 3:
6 * 4 = 24
Considering these steps: 24 = 6 * 4 = (14 - 8) * 4
= (14 - 8) * (8 / 2) = 24.
Answer: (14 - 8) * (8 / 2) = 24.

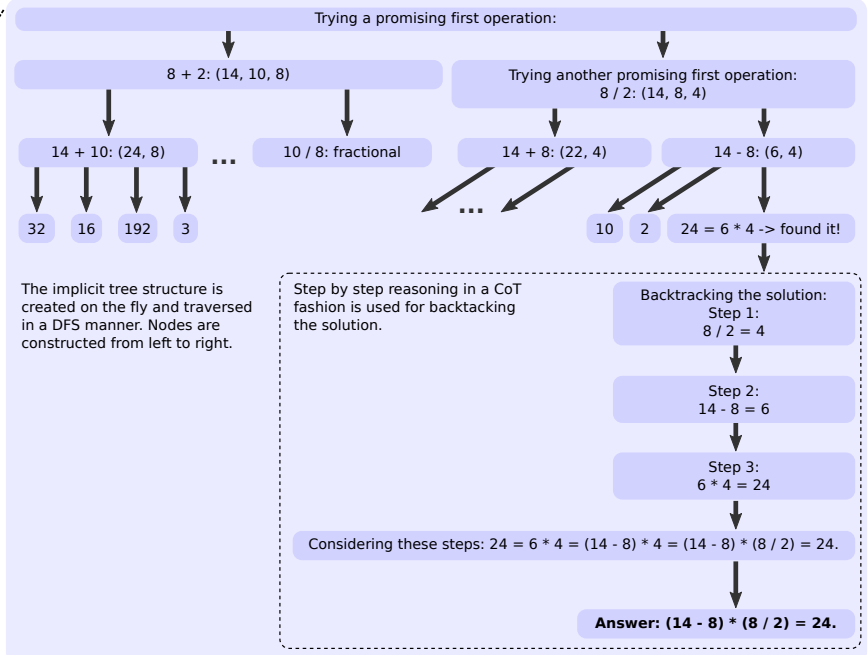**LLM Answer: Implicit Tree Structure**

Trying a promising first operation:

8 + 2: (14, 10, 8)   Trying another promising first operation: 8 / 2: (14, 8, 4)

14 + 10: (24, 8) ... 10 / 8: fractional   14 + 8: (22, 4)   14 - 8: (6, 4)

32 16 192 3   ...   10 2 24 = 6 * 4 -> found it!

The implicit tree structure is created on the fly and traversed in a DFS manner. Nodes are constructed from left to right.

Step by step reasoning in a CoT fashion is used for backtacking the solution.

Backtracking the solution:
Step 1:
8 / 2 = 4

Step 2:
14 - 8 = 6

Step 3:
6 * 4 = 24

Considering these steps: 24 = 6 * 4 = (14 - 8) * 4 = (14 - 8) * (8 / 2) = 24.

Answer: (14 - 8) * (8 / 2) = 24.

Fig. 10: GAME OF 24. **An example** implicit single-prompt **tree topology, encoded with** text**.** It demonstrates a Game of 24 DFS in-context example from AoT [166]. The left view shows the user prompt and the single textual answer from the LLM. The right view shows the implicit tree structure that is explored during the generation of the LLM answer. We mark text corresponding to implicit nodes as **bold**.

Creative Writing

User

Write a coherent passage of 4 short paragraphs. The end sentence of each paragraph must be: **1.** It isn't difficult to do a handstand if you just stand... **2.** ... **3.** ... **4.** ...

Plan 1
1. Introduce and explain the technique of doing a handstand.
2. ... **3.** ... **4.** ...

Plan 2
1. Introduction to an unusual self-help book, mentioning a handstand as a metaphor for embracing challenges.
2. ... **3.** ... **4.** ...

...
Plan 2 is chosen.

Passage 1   Passage 2   ...
Passage 2 is chosen.

Fig. 11: CREATIVE WRITING. **An example** explicit multi-prompt **tree topology, encoded with** text, from the Tree of Thoughts (ToT) scheme [213] for creative writing. Given the task of writing a coherent passage of four paragraphs ending in given sentences, first multiple plans (nodes) are generated and then ranked. In a next step, the best plan is used to generate multiple possible passages as outputs. Finally, the best ranked passage is the output of the ToT reasoning.

Math Reasoning

User

[in-context examples]
Q: Tobias is buying a new pair of shoes that costs $95. He has been saving up his money each month [...]?

Answer 1
Let D be the number of driveways Tobias shoveled. He has been saving up [...]
So the answer is 1.

Answer 2
Let D be the number of driveways Tobias shoveled. He earned $5 allowance [...]
So the answer is 5.

Answer 3
Let M be the number of lawns Tobias mows and D be the number of [...]
So the answer is 5.

Select final answer based on majority vote: **So the answer is 5.**

Fig. 12: MATH REASONING. **An example using** explicit multi-prompt **tree topology, encoded with** text. Given a math reasoning task, CoT-SC [190] is used to generate multiple answers and pick a final one based on majority vote. Each of the generated answers contains multiple CoT reasoning steps, depicted here in a single node.

the question to be answered by the LLM. From this node, possible reasoning paths are generated by the LLM. For each path, new nodes, i.e., intermediate reasoning steps, are generated by the LLM and are then used to grow the graph. **Graph-of-Thought** [215] describes a two-stage framework to answer multi-modal questions, i.e., textual questions accompanied by images. In the first stage, the model generates natural language rationales based on the input text, which provide additional context and knowledge to support answering the given question. This rationale generation is learned as part of the overall model pipeline. In the second stage, these rationales are then appended to

the initial question and passed again through the model to predict an answer. Other schemes in this class include **Cumulative Reasoning** [224], **Everything of Thoughts (XoT)** [57], **ControlLLM** [132], and **ResPrompt** [99].

7.3 Hypergraphs

Finally, we also consider a *hypergraph*, which generalizes a graph by enabling edges to connect arbitrary subsets of nodes instead of being links between just two nodes. We include hypergraphs in the taxonomy, because preliminary works already harness them for multi-modal prompting [212]. Here, **Hypergraph-of-Thought (HoT)** [212] is a multi-modal reasoning paradigm modeling the thought pro-
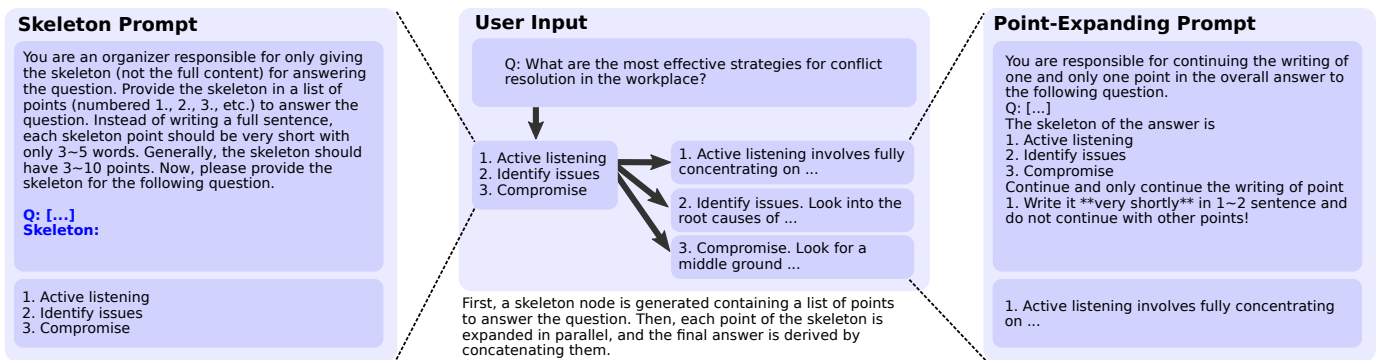
Question Answering

Fig. 13: **An example explicit multi-prompt tree topology, encoded with text**. It demonstrates the automatically derived tree topology of Skeleton-of-Thought (SoT) [148] where the individual points are expanded in parallel.

cess as a hypergraph. First, a graph-of-thoughts as in [215] is constructed. Then a textual hypergraph is constructed, sharing the same nodes. The hyperedges are then defined as node triples, e.g., "(Lionel Messi, place of birth, Rosario)". Additionally, a visual hypergraph-of-thought is constructed by performing k-means clustering on image patches, where a cluster corresponds to a hyperedge. Both hypergraphs are then encoded and combined to perform graph learning.

7.4 Analysis & Comparison of Designs

We now broadly discuss and analyze graph designs with respect to different aspects of our blueprint.

7.4.1 Topology & Its Construction

Firstly, the considered schemes exhibit a blend of single- and multi-prompt aspects, allowing for a high degree of flexibility and control over the prompting process. This is evident in the diverse approaches taken by different schemes such as GoT, ControlLLM, and Cumulative Reasoning, each offering unique ways of constructing and utilizing graphs for problem-solving. Secondly, the user's control over the topology of the graph is significant in most schemes, enabling customization of the reasoning process based on specific needs, such as setting branching factors or defining the depth of the graph. Thirdly, the role of the LLM in these graph-based schemes is multifaceted, involving the generation, evaluation, and modification of nodes within the graph, as well as determining the conclusion of the reasoning process. Lastly, there is a notable variation in the degree of user and LLM influence on the topology across different schemes, with some allowing direct user control, while others rely on predefined heuristics or the LLM's decision-making capabilities.

7.4.2 Representations of Topology & Schedule

We now illustrate a representative set of prompts that show different aspects of graph-based prompting topologies, focusing on how the respective graph topologies are encoded inside the prompts (the tasks are described in detail in the prompts of the figures). We illustrate ResPrompt and Cumulative Reasoning (as example implicit single-prompt representations), ControlLLM (as an example explicit single-prompt representation), and Branch-Solve-Merge (as an example multi-prompt representation).

Figure 14 shows a prompting example using Cumulative Reasoning [224] for the Game of 24 with an *explicit multi-prompt* graph topology. Figure 15 shows an in-context example of ResPrompt [99] with a multi-step math question, where the topology is *single-prompt* and *implicit*. Here, an (implicit) edge can be formed by repeating the same token, for example *"earned from his allowance"*, which implies a connection between step 2 and 4 where each step indicates a node. Figure 16 illustrates an in-context example where Cumulative Reasoning [224] is applied to the same math problem with user-specified number of intermediate nodes before reaching the final solution node. Here, *implicit* vertices are formed using numbered positions 1 and 2 on the list, and edges connect points 1 and 2 to point 3. In contrast to the implicit representation of topology, Figure 17 shows an example from ControlLLM [132] to represent the topology *explicitly* with a JSON format, but also in the *single-prompt* setting. Finally, Figure 18 depicts a *multi-prompt* example from Branch-Solve-Merge (BSM) [162] for story generation.

7.4.3 Performance

The considered works universally show improvements in effectiveness of graph-based prompting schemes over chains and trees across various tasks, suggesting a promising direction for future research and application in the field of AI and machine learning.

8 CHAINS VS. TREES VS. GRAPHS OF THOUGHTS

We also broadly discuss tradeoffs, commonalities, and differences between the three fundamental classes of topologies: chains, trees, and graphs.

The novelty in chain-based prompting lies in introducing explicit intermediate LLM thoughts between the input and the output. This linear sequence of thoughts guides the LLM in a step-by-step manner towards the solution, enhancing the clarity and traceability of the reasoning process. Beyond this, tree-based schemes bring the possibility to explore several next-step variants at each juncture, allowing the LLM to evaluate multiple pathways and select the most promising one. This branching structure facilitates a broader exploration of potential solutions. Graph-based schemes, however, represent the most complex structure, offering an
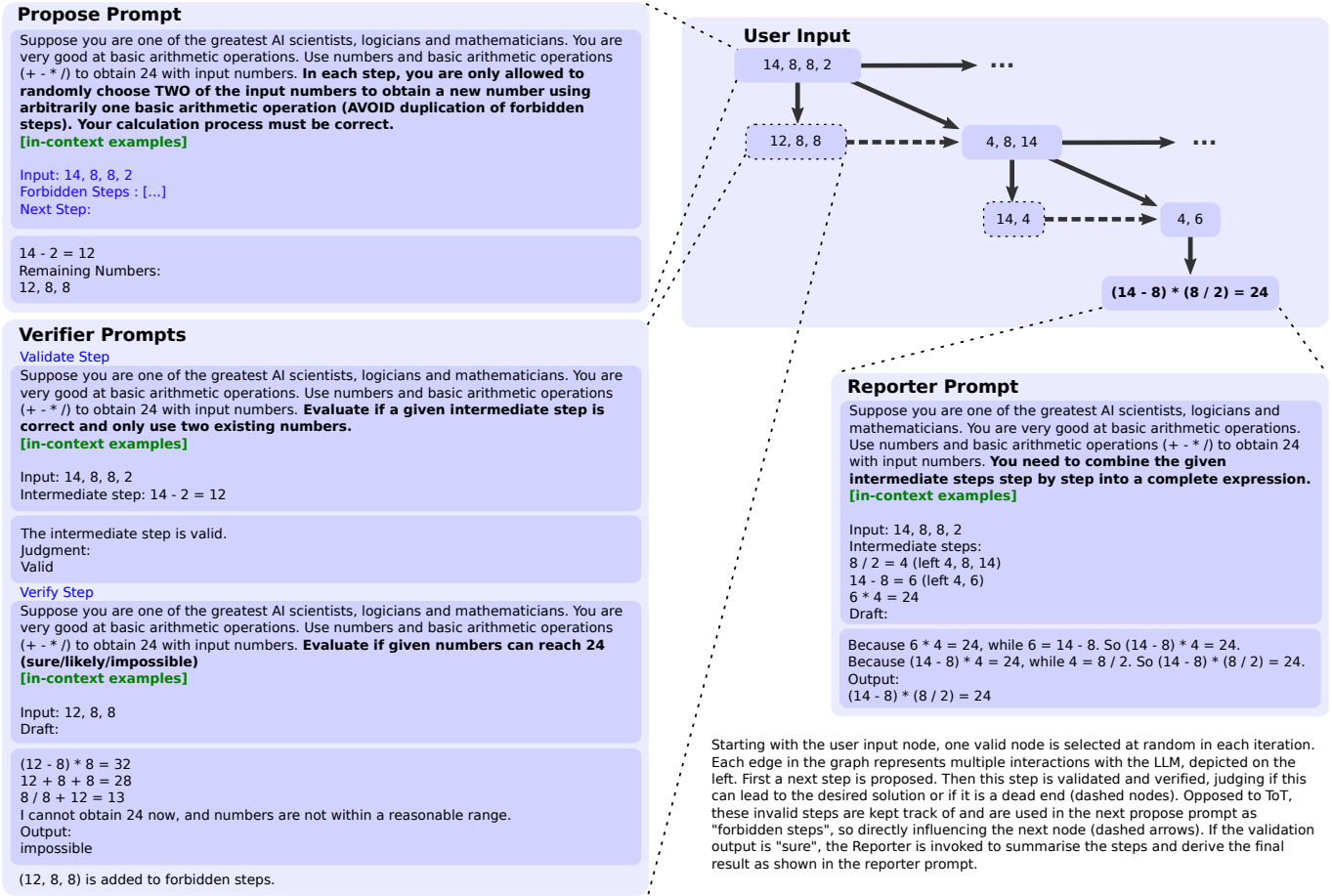
Game of 24

Propose Prompt

Suppose you are one of the greatest AI scientists, logicians and mathematicians. You are very good at basic arithmetic operations. Use numbers and basic arithmetic operations (+ - * /) to obtain 24 with input numbers. **In each step, you are only allowed to randomly choose TWO of the input numbers to obtain a new number using arbitrarily one basic arithmetic operation (AVOID duplication of forbidden steps). Your calculation process must be correct.**
[in-context examples]

Input: 14, 8, 8, 2
Forbidden Steps : [...]
Next Step:

14 - 2 = 12
Remaining Numbers:
12, 8, 8

Verifier Prompts

Validate Step

Suppose you are one of the greatest AI scientists, logicians and mathematicians. You are very good at basic arithmetic operations. Use numbers and basic arithmetic operations (+ - * /) to obtain 24 with input numbers. **Evaluate if a given intermediate step is correct and only use two existing numbers.**
[in-context examples]

Input: 14, 8, 8, 2
Intermediate step: 14 - 2 = 12

The intermediate step is valid.
Judgment:
Valid

Verify Step

Suppose you are one of the greatest AI scientists, logicians and mathematicians. You are very good at basic arithmetic operations. Use numbers and basic arithmetic operations (+ - * /) with input numbers. **Evaluate if given numbers can reach 24 (sure/likely/impossible)**
[in-context examples]

Input: 12, 8, 8
Draft:

(12 - 8) * 8 = 32
12 + 8 + 8 = 28
8 / 8 + 12 = 13
I cannot obtain 24 now, and numbers are not within a reasonable range.
Output:
impossible

(12, 8, 8) is added to forbidden steps.

User Input

14, 8, 8, 2 → ...

12, 8, 8 ⇢ 4, 8, 14 → ...

14, 4 ⇢ 4, 6

(14 - 8) * (8 / 2) = 24

Reporter Prompt

Suppose you are one of the greatest AI scientists, logicians and mathematicians. You are very good at basic arithmetic operations. Use numbers and basic arithmetic operations (+ - * /) to obtain 24 with input numbers. **You need to combine the given intermediate steps step by step into a complete expression.**
[in-context examples]

Input: 14, 8, 8, 2
Intermediate steps:
8 / 2 = 4 (left 4, 8, 14)
14 - 8 = 6 (left 4, 6)
6 * 4 = 24
Draft:

Because 6 * 4 = 24, while 6 = 14 - 8. So (14 - 8) * 4 = 24.
Because (14 - 8) * 4 = 24, while 4 = 8 / 2. So (14 - 8) * (8 / 2) = 24.
Output:
(14 - 8) * (8 / 2) = 24

Starting with the user input node, one valid node is selected at random in each iteration. Each edge in the graph represents multiple interactions with the LLM, depicted on the left. First a next step is proposed. Then this step is validated and verified, judging if this can lead to the desired solution or if it is a dead end (dashed nodes). Opposed to ToT, these invalid steps are kept track of and are used in the next propose prompt as "forbidden steps", so directly influencing the next node (dashed arrows). If the validation output is "sure", the Reporter is invoked to summarise the steps and derive the final result as shown in the reporter prompt.

Fig. 14: **GAME OF 24. An example** explicit multi-prompt **graph topology, encoded with** text using Cumulative Reasoning [224].

Math Reasoning

User Prompt

[in-context examples]
Q: Tobias is buying a new pair of shoes that costs $95. He has been saving up his money each month [...]?

LLM Answer

1. He has $15 left after buying the shoes that cost $95, which means he had $95 + $15 = $110 in total before the purchase.

2. Next, we calculate the amount he earned from his allowance. He has been saving for three months, getting $5 each month, [...]

3. Then we know he mowed 4 lawns and he charges $15 per lawn, so from mowing lawns, he earned 4 * $15 = $60.

4. Let us sum up the money he earned from his allowance and from mowing lawns: $15 (allowance) + $60 (mowing) = $75.

5. Now, let us subtract the total amount of money earned from allowance and mowing from his total amount of money before buying the shoes to [...].

6. As he charges $7 to shovel a driveway, we divide the total earned from shoveling by the charge per driveway to find out the number of driveways he shoveled: $35 / $7 = 5 driveways.

LLM Answer: Implicit Tree Structure

1. He has $15 left after buying the shoes that cost $95, which means he had $95 + $15 = $110 **in total before the purchase**.

2. Next, we calculate the amount he **earned from his allowance**. He has been saving for three months, getting $5 each month, [...]

3. Then we know he mowed 4 lawns and he charges $15 per lawn, so from mowing lawns, he earned 4 * $15 = $60.

4. Let us sum up the money he **earned from his allowance** and from mowing lawns: $15 (allowance) + $60 (mowing) = $75

5. Now, let us subtract the total amount of money earned from allowance and mowing from his **total amount of money before buying the shoes** to [...].

6. As he charges $7 to shovel a driveway, we divide the total earned from shoveling by the charge per driveway to find out the number of driveways he shoveled: $35 / $7 = 5 driveways.

Fig. 15: **MATH REASONING. An example** implicit single-prompt **graph topology, encoded with** text. It shows an in-context example of a math question from ResPrompt [99], representing a linear sequence of six connected nodes with two implicit edges of the graph topology, marked with two different colors (**red** and **blue**), together with their corresponding nodes.
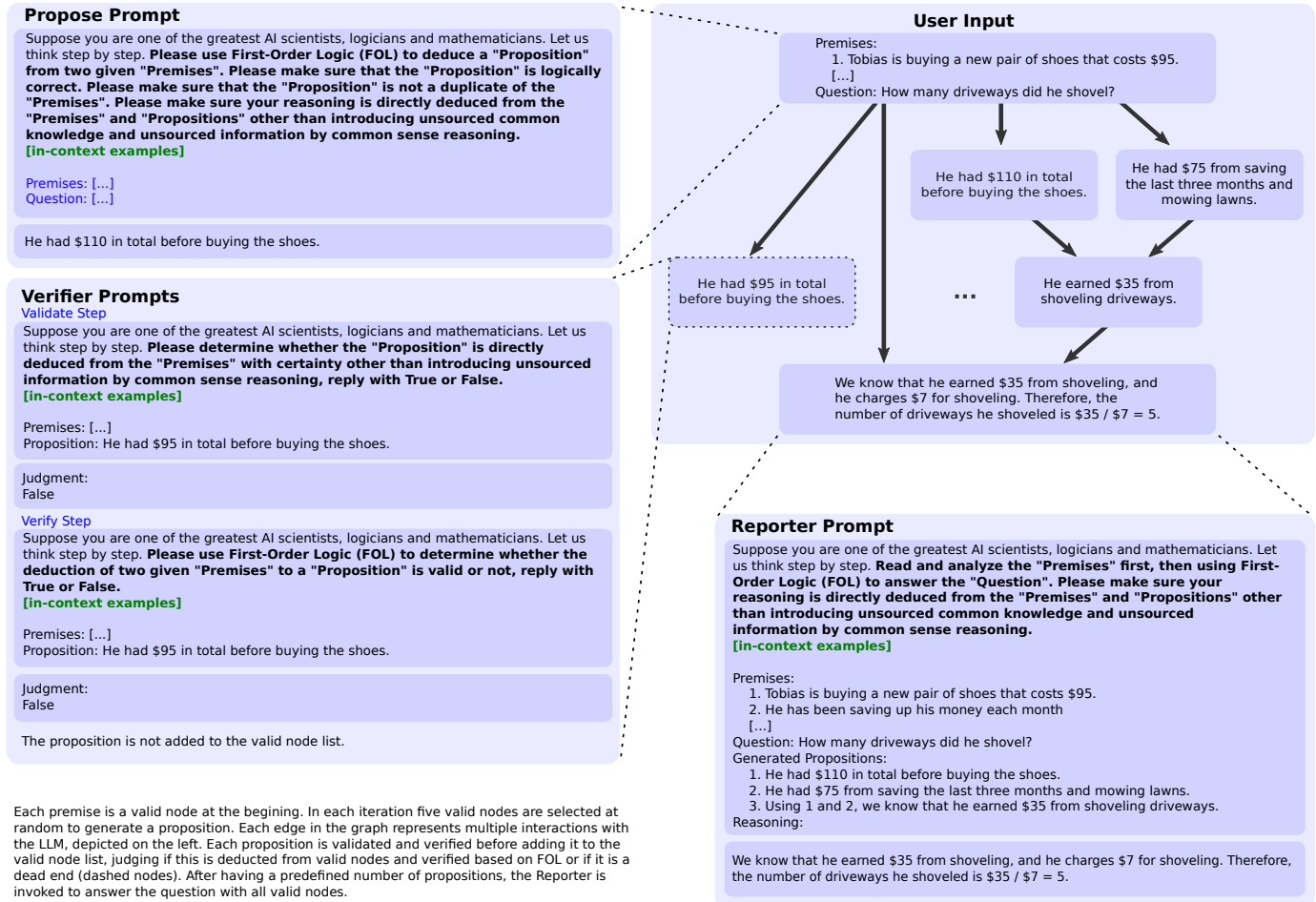
Math Reasoning

Propose Prompt

Suppose you are one of the greatest AI scientists, logicians and mathematicians. Let us think step by step. **Please use First-Order Logic (FOL) to deduce a "Proposition" from two given "Premises". Please make sure that the "Proposition" is logically correct. Please make sure that the "Proposition" is not a duplicate of the "Premises". Please make sure your reasoning is directly deduced from the "Premises" and "Propositions" other than introducing unsourced common knowledge and unsourced information by common sense reasoning.** **[in-context examples]**

Premises: [...]
Question: [...]

He had $110 in total before buying the shoes.

Verifier Prompts

Validate Step

Suppose you are one of the greatest AI scientists, logicians and mathematicians. Let us think step by step. **Please determine whether the "Proposition" is directly deduced from the "Premises" with certainty other than introducing unsourced information by common sense reasoning, reply with True or False.** **[in-context examples]**

Premises: [...]
Proposition: He had $95 in total before buying the shoes.

Judgment:
False

Verify Step

Suppose you are one of the greatest AI scientists, logicians and mathematicians. Let us think step by step. **Please use First-Order Logic (FOL) to determine whether the deduction of two given "Premises" to a "Proposition" is valid or not, reply with True or False.** **[in-context examples]**

Premises: [...]
Proposition: He had $95 in total before buying the shoes.

Judgment:
False

The proposition is not added to the valid node list.

Each premise is a valid node at the begining. In each iteration five valid nodes are selected at random to generate a proposition. Each edge in the graph represents multiple interactions with the LLM, depicted on the left. Each proposition is validated and verified before adding it to the valid node list, judging if this is deducted from valid nodes and verified based on FOL or if it is a dead end (dashed nodes). After having a predefined number of propositions, the Reporter is invoked to answer the question with all valid nodes.

User Input

Premises:
    1. Tobias is buying a new pair of shoes that costs $95.
    [...]
Question: How many driveways did he shovel?

He had $110 in total before buying the shoes.

He had $75 from saving the last three months and mowing lawns.

He had $95 in total before buying the shoes.

...

He earned $35 from shoveling driveways.

We know that he earned $35 from shoveling, and he charges $7 for shoveling. Therefore, the number of driveways he shoveled is $35 / $7 = 5.

Reporter Prompt

Suppose you are one of the greatest AI scientists, logicians and mathematicians. Let us think step by step. **Read and analyze the "Premises" first, then using First-Order Logic (FOL) to answer the "Question". Please make sure your reasoning is directly deduced from the "Premises" and "Propositions" other than introducing unsourced common knowledge and unsourced information by common sense reasoning.** **[in-context examples]**

Premises:
    1. Tobias is buying a new pair of shoes that costs $95.
    2. He has been saving up his money each month
    [...]
Question: How many driveways did he shovel?
Generated Propositions:
    1. He had $110 in total before buying the shoes.
    2. He had $75 from saving the last three months and mowing lawns.
    3. Using 1 and 2, we know that he earned $35 from shoveling driveways.
Reasoning:

We know that he earned $35 from shoveling, and he charges $7 for shoveling. Therefore, the number of driveways he shoveled is $35 / $7 = 5.

Fig. 16: **MATH REASONING. An example implicit single-prompt graph topology, encoded with text**, for the same question as in Figure 15, but using Cumulative Reasoning [224]. It illustrates three intermediate proposition nodes with direct connections from the first two nodes to the third one. The reasoning part indicates the final solution node, with direct connections from the third proposition and input nodes.

arbitrary reasoning framework. They enable the aggregation of various reasoning steps into a synergistic solution, allowing for non-linear and multifaceted problem-solving approaches. This diversity in structures reflects the varying needs and complexities of tasks that LLMs are expected to handle.

In terms of cost-effectiveness, chain-based prompting schemes generally outperform their tree-based and graph-based counterparts. This efficiency largely stems from their single-prompt nature, which requires fewer computational resources and less processing time. Tree and graph-based schemes, on the other hand, often involve multiple rounds of prompting to explore the various branches or connections in their respective structures, naturally incurring higher costs. However, this is not an inherent limitation of these schemes. Future developments in tree and graph-based prompting should aim to better encode the tree and graph structures within a single prompt. Such advancements could potentially combine the cost-effectiveness of single-prompt schemes with the enhanced quality and complexity management offered by tree and graph-based structures.

When it comes to the quality of outcomes, tree-based and graph-based prompting schemes typically surpass chain-based ones. This superior performance is attributed to their ability to explore a wider range of potential transformations at each step, going beyond the linear refinement seen in chains. Tree-based schemes, with their branching paths, offer multiple avenues for solution exploration at each decision point, leading to a more comprehensive search for the optimal outcome. Graph-based schemes take this a step further by allowing for an even more diverse set of connections and interactions between different steps in the reasoning process. This flexibility enables these schemes to handle more complex and multifaceted tasks effectively, leading to higher-quality outcomes in scenarios where simple linear reasoning is insufficient.

9 DESIGN ARCHITECTURES

We also analyze design tradeoffs. We consider the following aspects: design architecture (what the architecture-level decomposition of different schemes into interacting modules is), productivity & programmability (how well different schemes support efforts such as new extensions), and scalability as well as parallelizability (whether considered schemes can be parallelized and whether they scale well).

JSON
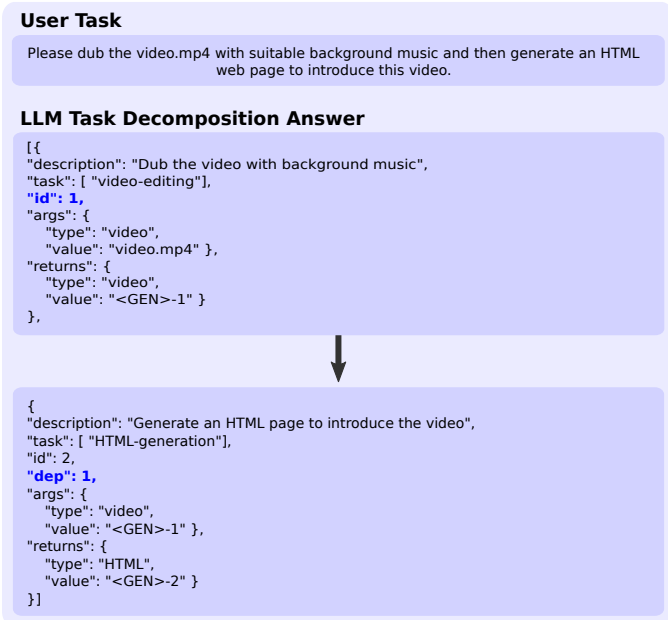
User Task

Please dub the video.mp4 with suitable background music and then generate an HTML web page to introduce this video.

LLM Task Decomposition Answer

```json
[{
"description": "Dub the video with background music",
"task": [ "video-editing"],
"id": 1,
"args": {
  "type": "video",
  "value": "video.mp4" },
"returns": {
  "type": "video",
  "value": "<GEN>-1" }
},
```

```json
{
"description": "Generate an HTML page to introduce the video",
"task": [ "HTML-generation"],
"id": 2,
"dep": 1,
"args": {
  "type": "video",
  "value": "<GEN>-1" },
"returns": {
  "type": "HTML",
  "value": "<GEN>-2" }
}]
```

Fig. 17: **JSON. An example explicit single-prompt graph topology, encoded with JSON**, based on the ControlLLM scheme [132] for task decomposition. It shows two nodes describing decomposed subtasks for solving a given task. The "dep" field refers to dependent tasks, showing there is a direct edge from the first node (task 1) to the second.

Creative Writing

User

shirt, dunk, bang, soap, court, float, swimmer, coat, sleigh, groom

Branch 1

shirt, dunk, bang, soap, court

[story topic]
a winter wedding adventure

Branch 2

float, swimmer, coat, sleigh, groom

Story 1

a loud bang echoed ..., dunking his shirt in a bucket of soap and water, and then using it to clean the court. ...

they were greeted by a swimmer, ... The groom, a skilled float operator, ... dressed in matching red coats, rode on a colorful sleigh, ...

Story 2

Generated Story

a loud bang echoed through the church, ... dunking his shirt in a bucket of soap and water, and then using it to clean the court. ... Meanwhile, outside, the groom, a skilled float operator, ... , rode on a colorful sleigh, ... a swimmer, ..., black coat, gracefully ...

Fig. 18: CREATIVE WRITING. **An example multi-prompt graph topology, encoded with text**, from the Branch-Solve-Merge scheme [162] for story generation with branch, solve, and merge prompts. Given a list of concepts as input, the branch module generates three child nodes: two groups of concepts and one topic node. The solve module then creates two story nodes based on each group of concepts and the topic. Finally, these two story nodes are merged into the final solution node.

9.1 Design Architecture

Here, we analyze how the overall prompting scheme, combining the LLM and some additional logic, decomposes into different modules. The module-level design architecture is explicitly discussed to a certain degree by some tree-based approaches, such as [50], [91], [133], [145], [213] as well as graph-based approaches such as [10], [119], [154], [162], [218], [224]. Detailed module-level architectures are presented by Long [133], Yao et al. [213], and Besta et al. [10].

The general architecture of considered schemes can be summarized as consisting of the following modules: a *gen-erator*, an *evaluator*, a *halter*, and a *controller* module. The generator prompts the LLM to produce further reasoning steps given the current reasoning step and some context. The evaluator rates the current reasoning step, depending on the implementation this rating can depend on other states, the path to the initial question node, or some other context. The halter determines if a suitable solution has been found and how it should be reported or if the search should continue. The controller module coordinates the other modules as well as the construction and exploration of the tree/graph.

9.2 Productivity & Programmability

ToT by Yao et al. [213], Tree Prompting [170] and SoT [148] provide implementations that can be used directly for custom tasks; ToT [213] is the only tree approach that provides an easily usable API to solve custom problems in a multi-prompt fashion. ToT by Long [133], Thought Decomposition [205] and ToC [106] provide implementations to reproduce their results, but can not easily be used for other tasks. CoT-SC [190] is a paradigm and can easily be implemented when needed, AoT [166] is a single-prompt scheme and does not provide an implementation but lists in-context examples. The remaining approaches [50], [58], [91], [145] have no code published at this time.

In graph designs, Socratic Questioning [154] and Graph of Thoughts (GoT) [10] provide implementations that can be used directly for custom tasks. Cumulative Reasoning [224] provides an implementation to reproduce their results, but cannot easily be used for other tasks. Thought Propagation (TP) [218], Branch-Solve-Merge (BSM) [162], ControlLLM [132] and ResPrompt [99] do not provide implementation but list a set of prompting examples. Among these, ResPrompt [99] is a paradigm can be easily implemented when needed. The remaining approaches [57], [119], [212], [215] have no code published at this time.

Overall, GoT [10] offers the most general API for addressing custom tasks in a multi-prompt fashion. It offers a principled way to design a prompting structure, through *thought transformations*, i.e., ways to transform GoT nodes into new outcomes. Example GoT API routines, which implement thought transformations include – for example *Generate(prompt, k=4)* (prompt the LLM to produce k answers to a given prompt), *Generate(prompt, k=1)+Repeat(k=4)* (generate 4 context isolated responses of the LLM), *Aggregate(thought1, thought2)* (combine two thoughts), *KeepBest(N=1)* (query the LLM to return N best results), *Improve(thought)* (query the LLM to improve the result using information from another thought, e.g., input list and currently sorted list $\rightarrow$ try to fix incorrectly sorted elements).

9.3 Scalability & Parallelizability

Only a few works address scalability and parallelizability improvements of LLMs. Skeleton of Thought [148] uses the ToT approach: the prompt query is a tree of depth one and all leaves can be processed **in parallel**. Batch Prompt [124] improves the performance by **batching** different data points into a single prompt. On the other hand, Lookahead Decoding [65] is focused on **speculatively** guessing tokens to

speed up the answer generation. We now discuss in more detail how these concepts are realized.

Skeleton of Thought [148] tackles the problem of high inference latency in LLMs by addressing the sequential decoding problem (generating answer tokens one at a time). They proposed a new approach where a *skeleton* prompt leads the LLM to first generate a succinct list of points that form the skeleton of its future answer. Then, each point in the skeleton is decoded in parallel, using batching whenever possible. The method does not require changes to LLMs and can be applied to off-the-shelf models. with reported speedups of up to 2.69×. However, this method ignores dependencies between points in the skeleton answer, and the authors propose replacing the tree with a graph model as future work.

Lookahead Decoding [65] extends the idea of *speculative decoding* where a sequence of future tokens is predicted, and the LLM later verifies each one of them in parallel. This method represents the sequential generation of a chain of responses as a non-linear system of equations and then solves it with the Jacobi iteration method. In each iteration step, at least one guessed token - the next one - is verified and matched successfully. Furthermore, the method caches *n-grams* generated for each token by inspecting N prior Jacobi iterations, allowing it to decode multiple tokens on a positive match. By conducting the lookahead and verification in parallel, authors achieve decoding speedup by up to 2.25×, at the cost of an exponential increase in GPU FLOPs.

BatchPrompt [124] is a scheme that batches similar questions into a single prompt to reduce the overhead of task description and few-shot examples compared to single-question prompting. However, naive batching decreases the overall LLM performance, and the final accuracy depends on the position of each single prompt within the batch, which may be challenging to tune. BatchPrompt avoids this problem by performing majority voting on multiple rounds of prompting with different permutations of the question, allowing for competitive results compared to single-question prompting. By using large batches with few voting rounds, the scheme can process queries with significantly fewer calls to the LLM.

10 FOUNDATIONS & THEORY

There are a few preliminary works that attempt to provide foundations for structured prompting.

Madaan and Yazdanbakhsh [141] focus on **dissecting CoT into three basic components**, namely *symbols* (sequences of tokens used as the basis of the LLM reasoning, for example <u>numbers to be sorted</u>), *patterns* (the structure within prompt that reinforces task understanding, for example the <u>order</u> of numbers to be sorted), and *text* (any tokens that are <u>neither</u> a symbol <u>nor</u> a pattern). Here, "patterns" is effectively a certain single-prompt topology that enhances the LLM reasoning. The authors discover that, for example, patterns are relevant for better task understanding, and they form a symbiotic relationship with text: the latter help to create more useful patterns while the former enable the LLM to generate text that helps in task solving.

Tutunov et al. [185] use **probabilistic graphical models** to provide a theoretical understanding of how and why LLMs are able to generate a coherent chain of thoughts. Their model generalizes Jiang's latent space theory of language models [98] to chains of thoughts. In the latent space theory, thoughts convey a hidden *intention*. Because the intention is not directly observed – only the uttered thought – there is a potential for ambiguity. This *ambiguity* ρ can be quantified probabilistically for the language as a whole. Tutonov et al. extend Jiang's model by adding an additional *hidden context* variable upon which intentions are conditioned. This context c defines a family of tasks which each define a set of coherent reasoning steps. Moreover, Tutonov et al. generalize the one-shot generation of thoughts into a chain of thoughts with an associated *chain of intentions*. Every intention is conditioned on the context c and the previous intentions. In turn, the i-th thought in the chain is conditioned on the i-th intention. Their main result is as follows: conditioned on the input and a sequence of N example chain of thoughts generated from a context c, the probability that an LLM assigns to a chain of thoughts approaches the true probability of the chain given the input and the hidden context c with convergence speed ρ^N. This means that, with the appropriate examples, LLMs can generate a chain of thoughts that is arbitrarily close to the true output.

Besta et al. [10] discuss **tradeoffs between latency** (number of steps to reach the final thought) **and volume**, which they define – for a given thought t – as *the number of preceding LLM thoughts that could have impacted thought t*. Formally, it is the number of thoughts in the topology from which there exists a path to thought t. They assume a time of $O(1)$ for each reasoning step, fix the total cost for each topology in their analysis to $\Theta(n)$ and also make certain assumptions for each topology. For a single chain both the latency and the volume have a high value of N, which for multiple chains (k independent chains) is reduced by k for both metrics (N/k). In their tree analysis they assume a complete k-ary tree, which has a low latency of $\lceil \log_k N \rceil$, but the volume is similarly low ($O(\log_k N)$). For graphs, they assume two complete k-ary trees, where the first tree, representing the division into sub-tasks, is joined at the leaf level with another "reversed" k-ary tree (including its edges), representing the aggregation into the final solution. They conclude that double-tree graphs provide the best tradeoff with a latency of $\log_k N$ and a volume of N.

Several works, while not providing any theoretical underpinning for structured-enhanced prompting, investigate the capabilities for CoT to conduct formal analyses. This includes GSM8K [48], ProofWriter [179], FOLIO [82], SimpleLogic [221], and PrOntoQA [164].

Finally, there have been several works that investigate the theoretical underpinning of general in-context learning. This includes Xie et al. [204], Wies et al. [200], Hahn and Goyal [80], and Jiang [98]. However, as they do not focus on the topologies of prompting, details are outside the focus of our work.

11 RESEARCH OPPORTUNITIES

We now review future research directions in structure-enhanced prompting.

Exploring New Topology Classes An intriguing related approach is the study of novel *topology classes*, such

as hypergraphs, in prompting schemes. The existing HoT scheme [212] primarily uses triples, barely scratching the surface of what hypergraphs can offer in terms of their connectivity structure, where hyperedges can connect arbitrary subsets of vertices. Future research could delve into how hypergraphs can accommodate more complex relationships in data such as motifs [18], dense subgraphs [30], cliques [25], [74], [174], and others, potentially leading to breakthroughs in LLM's understanding and reasoning capabilities.

Explicit Representations in Single-Prompt Settings The explicit representation of prompting topologies has been largely unexplored, especially in scenarios involving single prompts. Research in this area could focus on how different representations, such as Adjacency Lists, Adjacency Matrix, or numerous others [29], can be employed to enhance the efficiency and effectiveness of LLMs. This exploration is vital, considering the impact of different data structures on the processing and interpretation of information by LLMs. The challenge lies in integrating these complex representations into a single-prompt format without compromising the simplicity and accessibility of the models.

Automatic Derivation of Tree and Graph Topologies The majority of current tree and graph topologies in LLM prompting are created manually or semi-automatically. A promising research direction is the development of methodologies for the automatic derivation of these topologies. Automating this process could significantly reduce the time and effort required to configure LLMs for specific tasks, making them more accessible and efficient. This automation might involve leveraging machine learning algorithms to identify and implement the most effective topologies based on the nature of the task and the data involved.

Advancements in Single-Prompt Schemes While there are a few examples of single-prompt schemes, such as the initial Chain-of-Thought (CoT), there's much room for improvement. Future research could focus on how to encapsulate more complex and detailed tree and graph structures within a single prompt, because it could reduce computational costs and simplify user interaction with LLMs (as one does not need multiple prompting interactions to build a given topology). The challenge lies in balancing the richness of the prompt with the need to maintain clarity and avoid overwhelming the model. Here, one could harness a recent line of works related to encoding graph structures within a prompt, such as GPT4Graph [79], GraphText [229], GraphGPT [181], LLMs-as-Predictors [44], and others [36], [61], [92], [94], [127], [155], [186], [187], [216], [222], [223], [227].

Investigating New Scheduling Approaches Most current structure-enhanced prompting schemes rely on standard scheduling algorithms like Breadth-First Search (BFS), Depth-First Search (DFS), or are manually designed. Exploring new scheduling techniques could lead to more efficient and effective processing of prompts. This could include adaptive scheduling algorithms that adjust their approach based on the nature of the task or the responses of the LLM, potentially enhancing the model's performance in complex reasoning tasks.

Investigating Novel Graph Classes An interesting idea is to explore *graph classes* for more effective and more efficient reasoning topologies. While current approaches have

harnessed, among others, general directed graphs, utilizing specialized ones could lead to better reasoning routines. One could harness, for example, structures behind low-diameter networks [19], [20], [96], [107], [108], [116], [117], [118] for potential gains in theoretical properties of LLM reasoning, such as lower latency.

Integration with Graph Algorithms and Paradigms Integrating graph-related algorithms and paradigms could offer more powerful representations and schedules in LLM prompting. This integration might involve the use of advanced graph algorithms to optimize the structure and flow of the prompting process [11], [13], potentially leading to more accurate and efficient outcomes. Research could explore how such different graph paradigms [26] can be adapted to the unique requirements of LLM prompting.

Diversifying Modalities in Prompting Different modalities in prompting, such as visual, auditory, or kinesthetic, are currently underexplored. Research in this area could involve developing multi-modal prompting systems that can understand and respond to inputs in various forms while harnessing the advantages of graphs or trees of thoughts. This diversification could lead to more interactive and inclusive LLM systems that cater to a wider range of users and use cases.

Enhancing Retrieval in Prompting Retrieval in prompting is another area that has received certain attention, with various recent schemes [6], [7], [40], [68], [100], [106], [191], [199], [219]. Improving retrieval mechanisms could enhance the LLM's ability to access and utilize relevant information more efficiently. This might involve developing more sophisticated algorithms for data retrieval or integrating external databases and knowledge bases [4], [5], [15], [16], [53] to expand the scope and depth of the LLM's responses.

Parallel Design in Prompting The aspect of parallel design in runtime prompting is an area that remains mostly unaddressed, and only a few schemes such as Skeleton-of-Thought address this challenge. Building upon these attempts could significantly enhance the speed and efficiency of LLMs. Research could focus on developing models that can simultaneously process multiple components of a prompt or handle various tasks in parallel, or the appropriate mapping to massively parallel architectures [19], [28], [73], [95], thereby reducing latency and improving user experience. One could also investigate effective integration of prompting with distributed-memory infrastructure and paradigms, such as remote direct memory access (RDMA) [21], [55], [56], [70] or serverless processing [49].

Integrating Structure-Enhanced Prompting with Graph Neural Networks A potential area of exploration is the integration of structure-enhanced prompting with Graph Neural Networks (GNNs) [22], [27], [37], [111], [203], [225], [234] and other mechanisms for Graph Machine Learning [12], [33], [81]. GNNs, known for their proficiency in handling relational data and capturing dependencies in graph structures, could augment the capabilities of LLMs in processing complex, structured prompts. By embedding prompting structures into graph-based representations, GNNs can provide a more nuanced and context-aware interpretation of the prompts, potentially leading to richer and more accurate responses. Furthermore, leveraging graph-related embeddings [12], [75] can enhance the LLMs' ability to

capture the subtleties in the relationships and hierarchies present in the prompts. Finally, harnessing heterogeneous GNNs [139], [178], [220] for integration with the concept of different semantic roles of vertices could be an interesting direction.

Integrating Structure-Enhanced Prompting with Complex System Architectures An essential and emerging area of research is the integration of prompting capabilities into the environment of complex existing system architectures, such as graph representation learning systems [63], [121], [189], graph databases [14], [17], [23], or relational databases. This integration aims to facilitate direct and nuanced interactions with complex data structures stored in these systems. By embedding LLMs into these environments, the prompting process can leverage the inherent organizational and relational capabilities of these databases. This approach would allow LLMs to access, interpret, and manipulate large and intricate datasets more efficiently and accurately. For instance, integrating with graph databases could enable LLMs to naturally understand and utilize the connections and relationships within data, while relational databases could provide a structured and queryable data format that complements the LLMs' linguistic capabilities.

Hardware Acceleration Understanding energy and performance bottlenecks and mitigating them with specialized techniques such as processing-in-memory [1], [24], [72], [146], [147], [167], FPGAs [28], [54], [144], or even quantum devices [9] will likely be increasingly important. Such advances can also enable much more scalable models and model execution under stringent conditions.

12 RELATED WORK

We also broadly discuss related analyses, taxonomies, and surveys.

12.1 General Prompt Engineering

There exist several detailed overviews of general prompt engineering. The main difference is that we provide the first taxonomy and analysis of *structure-enhanced* prompting methods, in which we focus on the *topology of the LLM reasoning*. Wang et al. [194] provide an overview of interactive NLP, in which they outline interactions of LLMs and humans, knowledge bases, models/tools, and environments. Gu et al. [77] provide a systematic survey of prompt engineering on vision-language foundation models. Liu et al. [130] describe in great detail general prompting and describe this area through the "pre-train, prompt, and predict" paradigm of building a prompting scheme. Qiao et al. [156] overview prompting schemes related to reasoning. Chen et al. [38] review the potential of prompting with LLMs. Finally, Zhang et al. [228] and Chu et al. [47] review CoT and the associated prompting schemes.

12.2 Graph-Related Generative AI

There have also been numerous works on graphs and LLMs. The key difference is that these works focus mostly on *using graph structures as input data* (at pre-training, fine-tuning, or prompting stage). We instead focus on graphs

(and other structures) as *mechanisms enabling structured LLM reasoning* [93].

General overviews of using graphs together with LLMs have been outlines by Li et al. [122] and Zhang et al. [226]. Pan et al. [150] very briefly mention some of the methods for integrating graphs with LLMs. Zong et al. briefly mention graph-related fusion in their work on self-supervised multi-modal learning [236]. Yang et al. [208] analyze data-centric graph learning.

Next, various works propose to **enhance general generative models with knowledge graphs (KGs)**. The focus of these works is usually to use KGs in order to enhance the LLM answers, for example by grounding knowledge in general models to reduce effects such as hallucinations [90], [149], [196], [209], [210]. Example schemes include Knowledge Graph Prompting (KGP) [192], Graph Neural Prompting (GNP) [183], Think-on-Graph (ToG) [176], Knowledge Solver (KSL) [62], KnowledGPT [191], and others [32], [138]. Zhu et al. [235] discuss how LLMs can be used for enhancing KG construction and tasks. Wen et al. [197] present MindMap, a framework to perform reasoning on KG data. Pertinent triples from a KG are retrieved and the LLM is prompted to answer a question based on these triples and show the reasoning process by generating a "mind map" in the form of a textual reasoning tree.

Several works discuss **graph foundation models** [66], [129]. These works discuss how to – in analogy to LLMs – devise general models pre-trained on massive amounts of graph data, that could then be fine-tuned for more specific applications, and used together with prompting for answering various graph-related tasks.

Recents schemes illustrate how to run **analytics on graphs using prompting** [101]; examples include GPT4Graph [79], GraphText [229], GraphGPT [181], LLMs-as-Predictors [44], and others [36], [61], [92], [94], [127], [155], [186], [187], [216], [222], [223], [227].

Finally, several works [177], [202] describe techniques for **graph prompt learning**, which is a class of approaches for enhanced prompting with graph pre-trained models. This class of schemes is orthogonal to our work, because it focuses on prompting for graph-oriented tasks, and it does not target the LLM structure of reasoning.

13 CONCLUSION

In conclusion, the rise of Large Language Models (LLMs) has revolutionized machine learning, extending their applications beyond Natural Language Processing (NLP) into diverse fields like medicine, logical reasoning, and planning. Prompt engineering has emerged as a crucial area, democratizing access to LLMs and offering a cost-effective alternative to fine-tuning and pre-training. However, challenges arise in optimizing LLM queries for complex tasks due to the inherent limitations of generative Transformer models.

This paper addresses these challenges by introducing a blueprint and an accompanying taxonomy of prompting schemes, focusing on the underlying structure of reasoning. We propose to model a general prompting scheme as a graph topology, where different classes of graphs, such as k–ary trees or directed graphs, can be used to reflect the structure and gain insights into different prompting

schemes. The taxonomy is then used to survey and analyze existing designs, dissecting them into fundamental aspects such as the representation of the reasoning topology, the derivation of the topology, or the encoding of the reasoning schedule. This taxonomy forms a blueprint that can be used to facilitate designing more effective prompting schemes.

We also conduct an analysis of structure-enhanced prompting methods in terms of their accuracy and quality of outcomes, latency, and cost-effectiveness. Our investigation results in different insights into the tradeoffs between prompting schemes, which facilitate selecting the best method for a given budget or workload target.

Furthermore, we investigate the preliminary works into foundations of structured-enhancing prompting, parallel and scalable designs, and productivity and programmability. We also provide valuable insights into open challenges and potential research directions, navigating the path for future research avenues into more advanced prompting.

ACKNOWLEDGEMENTS

REFERENCES

[1] J. Ahn, S. Yoo, O. Mutlu, and K. Choi. PIM-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ISCA '15, pages 336–348, Portland, Oregon, 2015. Association for Computing Machinery.

[2] A. Amini, S. Gabriel, P. Lin, R. Koncel-Kedziorski, Y. Choi, and H. Hajishirzi. MathQA: Towards Interpretable Math Word Problem Solving with Operation-Based Formalisms,, 2019. arXiv:1905.13319.

[3] G. Angeli, M. J. Johnson Premkumar, and C. D. Manning. Leveraging Linguistic Structure For Open Domain Information Extraction. In C. Zong and M. Strube, editors, *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, ACL-IJCNLP '15, pages 344–354, Beijing, China, Jul 2015. Association for Computational Linguistics.

[4] R. Angles and C. Gutierrez. Survey of Graph Database Models. *ACM Comput. Surv.*, 40(1), Feb 2008.

[5] R. Angles and C. Gutierrez. An Introduction to Graph Data Management. In G. H. L. Fletcher, J. Hidders, and J. L. Larriba-Pey, editors, *Graph Data Management, Fundamental Issues and Recent Developments*, Data-Centric Systems and Applications, pages 1–32. Springer, 2018.

[6] D. Arora, A. Kini, S. R. Chowdhury, N. Natarajan, G. Sinha, and A. Sharma. GAR-meets-RAG Paradigm for Zero-Shot Information Retrieval, 2023. arXiv:2310.20158.

[7] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection, 2023. arXiv:2310.11511.

[8] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, and C. Sutton. Program Synthesis with Large Language Models, 2021. arXiv:2108.07732.

[9] K. Bertels, A. Sarkar, A. Krol, R. Budhrani, J. Samadi, E. Geoffroy, J. Matos, R. Abreu, G. Gielen, and I. Ashraf. Quantum Accelerator Stack: A Research Roadmap, 2021. arXiv:2102.02035.

[10] M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, L. Gianinazzi, J. Gajda, T. Lehmann, M. Podstawski, H. Niewiadomski, P. Nyczyk, and T. Hoefler. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. In *Proceedings of the Thirty-Eigth AAAI Conference on Artificial Intelligence*, AAAI '24, Vancouver, Canada, Feb 2024. AAAI Press.

[11] M. Besta, A. Carigiet, K. Janda, Z. Vonarburg-Shmaria, L. Gianinazzi, and T. Hoefler. High-Performance Parallel Graph Coloring with Strong Guarantees on Work, Depth, and Quality. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20, Atlanta, Georgia, 2020. IEEE Press.

[12] M. Besta, A. C. Catarino, L. Gianinazzi, N. Blach, P. Nyczyk, H. Niewiadomski, and T. Hoefler. HOT: Higher-Order Dynamic Graph Representation Learning with Efficient Transformers. In *Proceedings of the Learning on Graphs Conference*, LOG '23, Nov 2023.

[13] M. Besta, M. Fischer, T. Ben-Nun, D. Stanojevic, J. De Fine Licht, and T. Hoefler. Substream-Centric Maximum Matchings on FPGA. *ACM Trans. Reconfigurable Technol. Syst.*, 13(2), Apr 2020.

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

[15] M. Besta, R. Gerstenberger, N. Blach, M. Fischer, and T. Hoefler. GDI: A Graph Database Interface Standard. https://github.com/spcl/GDI-RMA, Nov 2023. (accessed Jan. 15, 2024).

[16] M. Besta, R. Gerstenberger, M. Fischer, M. Podstawski, N. Blach, B. Egeli, G. Mitenkov, W. Chlapek, M. Michalewicz, H. Niewiadomski, J. Müller, and T. Hoefler. 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, Denver, CO, USA, 2023. Association for Computing Machinery.

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

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

[19] M. Besta, S. M. Hassan, S. Yalamanchili, R. Ausavarungnirun, O. Mutlu, and T. Hoefler. Slim NoC: A Low-Diameter On-Chip Network Topology for High Energy Efficiency and Scalability. *SIGPLAN Not.*, 53(2):43–55, Mar 2018.

[20] M. Besta and T. Hoefler. Slim fly: a cost effective low-diameter network topology. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 348–359, New Orleans, Louisana, 2014. IEEE Press.

[21] M. Besta and T. Hoefler. Active Access: A Mechanism for High-Performance Distributed Data-Centric Computations. In *Proceedings of the 29th ACM on International Conference on Supercomputing*, ICS '15, pages 155–164, Newport Beach, California, USA, 2015. Association for Computing Machinery.

[22] M. Besta and T. Hoefler. Parallel and Distributed Graph Neural Networks: An In-Depth Concurrency Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–20, 2024.

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

[24] M. Besta, R. Kanakagiri, G. Kwaśniewski, R. Ausavarungnirun, J. Beránek, K. Kanellopoulos, K. Janda, Z. Vonarburg-Shmaria, L. Gianinazzi, I. Stefan, J. G. Luna, J. Golinowski, M. Copik, L. Kapp-Schwoerer, S. Di Girolamo, N. Blach, M. Konieczny, O. Mutlu, and T. Hoefler. SISA: Set-Centric Instruction Set Archi-

tecture for Graph Mining on Processing-in-Memory Systems. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '21, pages 282–297. Association for Computing Machinery, 2021.

[25] M. Besta, C. Miglioli, P. Sylos Labini, J. Tětek, P. Iff, R. Kanakagiri, S. Ashkboos, K. Janda, M. Podstawski, G. Kwaśniewski, N. Gleinig, F. Vella, O. Mutlu, and T. Hoefler. 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, Dallas, Texas, 2022. IEEE Press.

[26] M. Besta, M. Podstawski, L. Groner, E. Solomonik, and T. Hoefler. To Push or To Pull: On Reducing Communication and Synchronization in Graph Computations. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '17, pages 93–104, Washington, DC, USA, 2017. Association for Computing Machinery.

[27] M. Besta, P. Renc, R. Gerstenberger, P. Sylos Labini, A. Ziogas, T. Chen, L. Gianinazzi, F. Scheidl, K. Szenes, A. Carigiet, P. Iff, G. Kwaśniewski, R. Kanakagiri, C. Ge, S. Jaeger, J. Was, F. Vella, and T. Hoefler. High-Performance and Programmable Attentional Graph Neural Networks with Global Tensor Formulations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, Denver, CO, USA, 2023. Association for Computing Machinery.

[28] M. Besta, D. Stanojevic, J. De Fine Licht, T. Ben-Nun, and T. Hoefler. Graph Processing on FPGAs: Taxonomy, Survey, Challenges, 2019. arXiv:1903.06697.

[29] M. Besta, D. Stanojevic, T. Zivic, J. Singh, M. Hoerold, and T. Hoefler. Log(Graph): A near-Optimal High-Performance Graph Representation. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, PACT '18, Limassol, Cyprus, 2018. Association for Computing Machinery.

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

[31] A. Bonifati, G. Fletcher, H. Voigt, and N. Yakovets. Data Models. In *Querying Graphs*, Synthesis Lectures on Data Management (SDLM), pages 3–14. Springer International Publishing, 2018.

[32] R. Brate, M.-H. Dang, F. Hoppe, Y. He, A. Meroño-Peñuela, and V. Sadashivaiah. Improving Language Model Predictions via Prompts Enriched with Knowledge Graphs. In *Proceedings of the Workshop on Deep Learning for Knowledge Graphs*, DL4KG@ISWC '22, Oct 2022.

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

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

[35] L. Cao. Enhancing Reasoning Capabilities of Large Language Models: A Graph-Based Verification Approach, 2023. arXiv:2308.09267.

[36] Z. Chai, T. Zhang, L. Wu, K. Han, X. Hu, X. Huang, and Y. Yang. GraphLLM: Boosting Graph Reasoning Ability of Large Language Model, 2023. arXiv:2310.05845.

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

[38] B. Chen, Z. Zhang, N. Langrené, and S. Zhu. Unleashing the potential of prompt engineering in Large Language Models: a comprehensive review, 2023. arXiv:2310.14735.

[39] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating Large Language Models Trained on Code, 2021. arXiv:2107.03374.

[40] W. Chen, H. Hu, X. Chen, P. Verga, and W. W. Cohen. MuRAG: Multimodal Retrieval-Augmented Generator for Open Question Answering over Images and Text, 2022. arXiv:2210.02928.

[41] W. Chen, X. Ma, X. Wang, and W. W. Cohen. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *Transactions on Machine Learning Research*, Nov 2023.

[42] Z. Chen, W. Chen, C. Smiley, S. Shah, I. Borova, D. Langdon, R. Moussa, M. Beane, T.-H. Huang, B. Routledge, and W. Y. Wang. FinQA: A Dataset of Numerical Reasoning over Financial Data. In M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, EMNLP '21, pages 3697–3711, Punta Cana, Dominican Republic, Nov 2021. Association for Computational Linguistics.

[43] Z. Chen, S. Li, C. Smiley, Z. Ma, S. Shah, and W. Y. Wang. ConvFinQA: Exploring the Chain of Numerical Reasoning in Conversational Finance Question Answering. In Y. Goldberg, Z. Kozareva, and Y. Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, EMNLP '22, pages 6279–6292, Abu Dhabi, United Arab Emirates, Dec 2022. Association for Computational Linguistics.

[44] Z. Chen, H. Mao, H. Li, W. Jin, H. Wen, X. Wei, S. Wang, D. Yin, W. Fan, H. Liu, and J. Tang. Exploring the Potential of Large Language Models (LLMs) in Learning on Graphs, 2023. arXiv:2307.03393.

[45] Z. Chen, K. Zhou, B. Zhang, Z. Gong, W. X. Zhao, and J.-R. Wen. ChatCoT: Tool-Augmented Chain-of-Thought Reasoning on Chat-based Large Language Models. In H. Bouamor, J. Pino, and K. Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 14777–14790, Singapore, Dec 2023. Association for Computational Linguistics.

[46] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez, I. Stoica, and E. P. Xing. Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90% ChatGPT Quality. https://lmsys.org/blog/2023-03-30-vicuna/, Mar 2023. (accessed Dec. 15, 2023).

[47] Z. Chu, J. Chen, Q. Chen, W. Yu, T. He, H. Wang, W. Peng, M. Liu, B. Qin, and T. Liu. A Survey of Chain of Thought Reasoning: Advances, Frontiers and Future, 2023. arXiv:2309.15402.

[48] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training Verifiers to Solve Math Word Problems, 2021. arXiv:2110.14168.

[49] M. Copik, G. Kwaśniewski, M. Besta, M. Podstawski, and T. Hoefler. SeBS: A Serverless Benchmark Suite for Function-as-a-Service Computing. In *Proceedings of the 22nd International Middleware Conference*, Middleware '21, pages 64–78, Québec City, Canada, 2021. Association for Computing Machinery.

[50] A. Creswell and M. Shanahan. Faithful Reasoning Using Large Language Models, 2022. arXiv:2208.14271.

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

[52] B. Dalvi, P. Jansen, O. Tafjord, Z. Xie, H. Smith, L. Pipatanangkura, and P. Clark. Explaining Answers with Entailment Trees. In M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, EMNLP '21, pages 7358–7370, Punta Cana, Dominican Republic, Nov 2021. Association for Computational Linguistics.

[53] A. Davoudian, L. Chen, and M. Liu. A Survey on NoSQL Stores. *ACM Comput. Surv.*, 51(2), Apr 2018.

[54] J. De Fine Licht, M. Besta, S. Meierhans, and T. Hoefler. Transformations of High-Level Synthesis Codes for High-Performance Computing. *IEEE Transactions on Parallel and Distributed Systems*, 32(5):1014–1029, May 2020.

[55] S. Di Girolamo, D. De Sensi, K. Taranov, M. Malesevic, M. Besta, T. Schneider, S. Kistler, and T. Hoefler. Building Blocks for

Network-Accelerated Distributed File Systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '22, Dallas, Texas, 2022. IEEE Press.

[56] S. Di Girolamo, K. Taranov, A. Kurth, M. Schaffner, T. Schneider, J. Beránek, M. Besta, L. Benini, D. Roweth, and T. Hoefler. Network-Accelerated Non-Contiguous Memory Transfers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, Denver, Colorado, 2019. Association for Computing Machinery.

[57] R. Ding, C. Zhang, L. Wang, Y. Xu, M. Ma, W. Zhang, S. Qin, S. Rajmohan, Q. Lin, and D. Zhang. Everything of Thoughts: Defying the Law of Penrose Triangle for Thought Generation, 2023. arXiv:2311.04254.

[58] A. Drozdov, N. Schärli, E. Akyürek, N. Scales, X. Song, X. Chen, O. Bousquet, and D. Zhou. Compositional Semantic Parsing with Large Language Models, 2022. arXiv:2209.15003.

[59] D. Dua, S. Gupta, S. Singh, and M. Gardner. Successive Prompting for Decomposing Complex Questions. In Y. Goldberg, Z. Kozareva, and Y. Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, EMNLP '22, pages 1251–1265, Abu Dhabi, United Arab Emirates, Dec 2022. Association for Computational Linguistics.

[60] D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, and M. Gardner. DROP: A Reading Comprehension Benchmark Requiring Discrete Reasoning Over Paragraphs. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, NAACL '19, pages 2368–2378, Minneapolis, Minnesota, Jun 2019. Association for Computational Linguistics.

[61] B. Fatemi, J. Halcrow, and B. Perozzi. Talk like a Graph: Encoding Graphs for Large Language Models, 2023. arXiv:2310.04560.

[62] C. Feng, X. Zhang, and Z. Fei. Knowledge Solver: Teaching LLMs to Search for Domain Knowledge from Knowledge Graphs, 2023. arXiv:2309.03118.

[63] M. Fey and J. E. Lenssen. Fast Graph Representation Learning with PyTorch Geometric. In *Proceedings of the Representation Learning on Graphs and Manifolds Workshop*, RLGM@ICLR '19, 2019.

[64] Y. Freund and R. E. Schapire. Experiments with a New Boosting Algorithm. In *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ICML '96, pages 148–156, Bari, Italy, 1996. Morgan Kaufmann.

[65] Y. Fu, P. Bailis, I. Stoica, and H. Zhang. Breaking the Sequential Dependency of LLM Inference Using Lookahead Decoding. https://lmsys.org/blog/2023-11-21-lookahead-decoding/, Nov 2023. (accessed Dec. 23, 2023).

[66] M. Galkin, X. Yuan, H. Mostafa, J. Tang, and Z. Zhu. Towards Foundation Models for Knowledge Graph Reasoning, 2023. arXiv:2310.04562.

[67] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig. PAL: Program-aided Language Models, 2022. arXiv:2211.10435.

[68] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, and H. Wang. Retrieval-Augmented Generation for Large Language Models: A Survey, 2023. arXiv:2312.10997.

[69] N. Garcia, C. Ye, Z. Liu, Q. Hu, M. Otani, C. Chu, Y. Nakashima, and T. Mitamura. A Dataset and Baselines for Visual Question Answering on Art. In A. Bartoli and A. Fusiello, editors, *Computer Vision – ECCV 2020 Workshops*, volume 12536 of *Lecture Notes in Computer Science*, pages 92–108. Springer, Jan 2020.

[70] R. Gerstenberger, M. Besta, and T. Hoefler. Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, Denver, Colorado, 2013. Association for Computing Machinery.

[71] M. Geva, D. Khashabi, E. Segal, T. Khot, D. Roth, and J. Berant. Did Aristotle Use a Laptop? A Question Answering Benchmark with Implicit Reasoning Strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361, 2021.

[72] S. Ghose, A. Boroumand, J. S. Kim, J. Gómez-Luna, and O. Mutlu. Processing-in-Memory: A Workload-driven Perspective. *IBM Journal of Research and Development*, 63(6):3:1–3:19, Nov 2019.

[73] L. Gianinazzi, T. Ben-Nun, M. Besta, S. Ashkboos, Y. Baumann, P. Luczynski, and T. Hoefler. The spatial computer: A model for energy-efficient parallel computation, 2022. arXiv:2205.04934.

[74] L. Gianinazzi, M. Besta, Y. Schaffner, and T. Hoefler. Parallel Algorithms for Finding Large Cliques in Sparse Graphs. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '21, pages 243–253. Association for Computing Machinery, 2021.

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

[76] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. Parikh. Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering, 2016. arXiv:1612.00837.

[77] J. Gu, Z. Han, S. Chen, A. Beirami, B. He, G. Zhang, R. Liao, Y. Qin, V. Tresp, and P. Torr. A Systematic Survey of Prompt Engineering on Vision-Language Foundation Models, 2023. arXiv:2307.12980.

[78] J. Gu, E. Stefani, Q. Wu, J. Thomason, and X. Wang. Vision-and-Language Navigation: A Survey of Tasks, Methods, and Future Directions. In S. Muresan, P. Nakov, and A. Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL '22, pages 7606–7623, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[79] J. Guo, L. Du, and H. Liu. GPT4Graph: Can Large Language Models Understand Graph Structured Data? An Empirical Evaluation and Benchmarking, 2023. arXiv:2305.15066.

[80] M. Hahn and N. Goyal. A Theory of Emergent In-Context Learning as Implicit Structure Induction, 2023. arXiv:2303.07971.

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

[82] S. Han, H. Schoelkopf, Y. Zhao, Z. Qi, M. Riddell, L. Benson, L. Sun, E. Zubova, Y. Qiao, M. Burtell, D. Peng, J. Fan, Y. Liu, B. Wong, M. Sailor, A. Ni, L. Nan, J. Kasai, T. Yu, R. Zhang, S. Joty, A. R. Fabbri, W. Kryscinski, X. V. Lin, C. Xiong, and D. Radev. FOLIO: Natural Language Reasoning with First-Order Logic, 2022. arXiv:2209.00840.

[83] M. Hartmann and D. Sonntag. A survey on improving NLP models with human explanations. In J. Andreas, K. Narasimhan, and A. Nematzadeh, editors, *Proceedings of the First Workshop on Learning with Natural Language Supervision*, pages 40–47, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[84] J. He-Yueya, G. Poesia, R. E. Wang, and N. D. Goodman. Solving Math Word Problems by Combining Language Models With Symbolic Solvers, 2023. arXiv:2304.09102.

[85] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring Massive Multitask Language Understanding. In *Proceedings of the International Conference on Learning Representations*, ICLR '21, 2021.

[86] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring Mathematical Problem Solving With the MATH Dataset. In *Proceedings of the Thirty-fifth Conference on Neural Information Processing Systems: Datasets and Benchmarks Track*, NeurIPS '21, 2021.

[87] X. Ho, A.-K. Duong Nguyen, S. Sugawara, and A. Aizawa. Constructing A Multi-hop QA Dataset for Comprehensive Evaluation of Reasoning Steps. In D. Scott, N. Bel, and C. Zong, editors, *Proceedings of the 28th International Conference on Computational Linguistics*, COLING '20, pages 6609–6625, Barcelona, Spain, Dec 2020. International Committee on Computational Linguistics.

[88] M. J. Hosseini, H. Hajishirzi, O. Etzioni, and N. Kushman. Learning to Solve Arithmetic Word Problems with Verb Categorization. In A. Moschitti, B. Pang, and W. Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, EMNLP '14, pages 523–533, Doha, Qatar, Oct 2014. Association for Computational Linguistics.

[89] H. Hu, H. Lu, H. Zhang, W. Lam, and Y. Zhang. Chain-of-Symbol Prompting Elicits Planning in Large Langauge Models, 2023. arXiv:2305.10276.

[90] L. Hu, Z. Liu, Z. Zhao, L. Hou, L. Nie, and J. Li. A Survey of Knowledge Enhanced Pre-Trained Language Models. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–19, 2023.

[91] P. Hu, J. Qi, X. Li, H. Li, X. Wang, B. Quan, R. Wang, and Y. Zhou. Tree-of-Mixed-Thought: Combining Fast and Slow Thinking for Multi-hop Visual Reasoning, 2023. arXiv:2308.09658.

[92] Y. Hu, Z. Zhang, and L. Zhao. Beyond Text: A Deep Dive into Large Language Models' Ability on Understanding Graph Data, 2023. arXiv:2310.04944.

[93] J. Huang and K. C.-C. Chang. Towards Reasoning in Large Language Models: A Survey. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 1049–1065, Toronto, Canada, Jul 2023. Association for Computational Linguistics.

[94] J. Huang, X. Zhang, Q. Mei, and J. Ma. Can LLMs Effectively Leverage Graph Structural Information: When and Why, 2023. arXiv:2309.16595.

[95] P. Iff, M. Besta, M. Cavalcante, T. Fischer, L. Benini, and T. Hoefler. HexaMesh: Scaling to Hundreds of Chiplets with an Optimized Chiplet Arrangement. In *Proceedings of the 60th ACM/IEEE Design Automation Conference*, DAC '23, pages 1–6, San Francisco, CA, USA, Jul 2023.

[96] P. Iff, M. Besta, M. Cavalcante, T. Fischer, L. Benini, and T. Hoefler. Sparse Hamming Graph: A Customizable Network-on-Chip Topology. In *Proceedings of the 2023 60th ACM/IEEE Design Automation Conference*, DAC '23, pages 1–6, San Francisco, CA, USA, Jul 2023.

[97] Jesse Dodge and Andreea Gane and Xiang Zhang and Antoine Bordes and Sumit Chopra and Alexander Miller and Arthur Szlam and Jason Weston. Evaluating Prerequisite Qualities for Learning End-to-End Dialog Systems, 2016. arXiv:1511.06931.

[98] H. Jiang. A Latent Space Theory for Emergent Abilities in Large Language Models, 2023. arXiv:2304.09960.

[99] S. Jiang, Z. Shakeri, A. Chan, M. Sanjabi, H. Firooz, Y. Xia, B. Akyildiz, Y. Sun, J. Li, Q. Wang, and A. Celikyilmaz. Resprompt: Residual Connection Prompting Advances Multi-Step Reasoning in Large Language Models, 2023. arXiv:2310.04743.

[100] Z. Jiang, F. Xu, L. Gao, Z. Sun, Q. Liu, J. Dwivedi-Yu, Y. Yang, J. Callan, and G. Neubig. Active Retrieval Augmented Generation. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, EMNLP '23, pages 7969–7992, Singapore, Dec 2023. Association for Computational Linguistics.

[101] B. Jin, G. Liu, C. Han, M. Jiang, H. Ji, and J. Han. Large Language Models on Graphs: A Comprehensive Survey, 2023. arXiv:2312.02783.

[102] J. Jung, L. Qin, S. Welleck, F. Brahman, C. Bhagavatula, R. Le Bras, and Y. Choi. Maieutic Prompting: Logically Consistent Reasoning with Recursive Explanations. In Y. Goldberg, Z. Kozareva, and Y. Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, EMNLP '22, pages 1266–1279, Abu Dhabi, United Arab Emirates, Dec 2022. Association for Computational Linguistics.

[103] D. Keysers, N. Schärli, N. Scales, H. Buisman, D. Furrer, S. Kashubin, N. Momchev, D. Sinopalnikov, L. Stafiniak, T. Tihon, D. Tsarkov, X. Wang, M. van Zee, and O. Bousquet. Measuring Compositional Generalization: A Comprehensive Method on Realistic Data. In *Proceedings of the Eigth International Conference on Learning Representations*, ICLR '20, Apr 2020.

[104] T. Khot, K. Richardson, D. Khashabi, and A. Sabharwal. Hey AI, Can You Solve Complex Tasks by Talking to Agents? In S. Muresan, P. Nakov, and A. Villavicencio, editors, *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1808–1823, Dublin, Ireland, May 2022. Association for Computational Linguistics.

[105] T. Khot, H. Trivedi, M. Finlayson, Y. Fu, K. Richardson, P. Clark, and A. Sabharwal. Decomposed Prompting: A Modular Approach for Solving Complex Tasks. In *Proceedings of the Eleventh International Conference on Learning Representations*, ICLR '23, May 2023.

[106] G. Kim, S. Kim, B. Jeon, J. Park, and J. Kang. Tree of Clarifications: Answering Ambiguous Questions with Retrieval-Augmented Large Language Models. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, EMNLP '23, pages 996–1009, Singapore, Dec 2023. Association for Computational Linguistics.

[107] J. Kim, W. J. Dally, and D. Abts. Flattened Butterfly: A Cost-efficient Topology for High-radix Networks. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 126–137, San Diego, California, USA, 2007. Association for Computing Machinery.

[108] J. Kim, W. J. Dally, S. Scott, and D. Abts. Technology-Driven, Highly-Scalable Dragonfly Topology. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, pages 77–88, Beijing, China, 2008. IEEE Computer Society.

[109] N. Kim and T. Linzen. COGS: A Compositional Generalization Challenge Based on Semantic Interpretation. In B. Webber, T. Cohn, Y. He, and Y. Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, EMNLP '20, pages 9087–9105. Association for Computational Linguistics, Nov 2020.

[110] S. Kim, S. Moon, R. Tabrizi, N. Lee, M. W. Mahoney, K. Keutzer, and A. Gholami. An LLM Compiler for Parallel Function Calling, 2023. arXiv:2312.04511.

[111] T. N. Kipf and M. Welling. Semi-Supervised Classification with Graph Convolutional Network. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17, Apr 2017.

[112] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large Language Models are Zero-Shot Reasoners. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems (NeurIPS '22)*, volume 35, pages 22199–22213. Curran Associates, 2022.

[113] R. Koncel-Kedziorski, S. Roy, A. Amini, N. Kushman, and H. Hajishirzi. MAWPS: A Math Word Problem Repository. In K. Knight, A. Nenkova, and O. Rambow, editors, *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL-HLT '16, pages 1152–1157, San Diego, California, Jun 2016. Association for Computational Linguistics.

[114] D. Kumar, V. Gupta, S. Sharma, and S. Zhang. Realistic Data Augmentation Framework for Enhancing Tabular Reasoning. In Y. Goldberg, Z. Kozareva, and Y. Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 4411–4429, Abu Dhabi, United Arab Emirates, Dec 2022. Association for Computational Linguistics.

[115] B. Lake and M. Baroni. Generalization without Systematicity: On the Compositional Skills of Sequence-to-Sequence Recurrent Networks. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on International Conference on Machine Learning (ICML '18)*, volume 80 of *Proceedings of Machine Learning Research*, pages 2873–2882. PMLR, Jul 2018.

[116] K. Lakhotia, M. Besta, L. Monroe, K. Isham, P. Iff, T. Hoefler, and F. Petrini. PolarFly: a cost-effective and flexible low-diameter topology. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '22, Dallas, Texas, 2022. IEEE Press.

[117] K. Lakhotia, K. Isham, L. Monroe, M. Besta, T. Hoefler, and F. Petrini. In-network Allreduce with Multiple Spanning Trees on PolarFly. In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '23, pages 165–176, Orlando, FL, USA, 2023. Association for Computing Machinery.

[118] K. Lakhotia, L. Monroe, K. Isham, M. Besta, N. Blach, T. Hoefler, and F. Petrini. PolarStar: Expanding the Scalability Horizon of Diameter-3 Networks, 2023. arXiv:2302.07217.

[119] B. Lei, p.-H. Lin, C. Liao, and C. Ding. Boosting Logical Reasoning in Large Language Models through a New Framework: The Graph of Thought, 2023. arXiv:2308.08614.

[120] Leo Breiman, Jerome Friedman, Charles J. Stone, R.A. Olshen. *Classification and Regression Trees*. Chapman and Hall, 1984.

[121] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, and S. Chintala. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. *Proc. VLDB Endow.*, 13(12):3005–3018, Aug 2020.

[122] Y. Li, Z. Li, P. Wang, J. Li, X. Sun, H. Cheng, and J. X. Yu. A Survey of Graph Meets Large Language Model: Progress and Future Directions, 2023. arXiv:2311.12399.

[123] Y. Li, Z. Lin, S. Zhang, Q. Fu, B. Chen, J.-G. Lou, and W. Chen. Making Language Models Better Reasoners with Step-Aware Verifier. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL '23, pages 5315–5333, Toronto, Canada, Jul 2023. Association for Computational Linguistics.

[124] J. Lin, M. Diesendruck, L. Du, and R. Abraham. BatchPrompt: Accomplish more with less, 2023. arXiv:2309.00384.

[125] W. Ling, D. Yogatama, C. Dyer, and P. Blunsom. Program Induction by Rationale Generation: Learning to Solve and Explain Algebraic Word Problems. In R. Barzilay and M.-Y. Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL '17, pages 158–167, Vancouver, Canada, Jul 2017. Association for Computational Linguistics.

[126] Z. Ling, Y. Fang, X. Li, Z. Huang, M. Lee, R. Memisevic, and H. Su. Deductive Verification of Chain-of-Thought Reasoning. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems (NeurIPS '23)*, volume 36, pages 36407–36433. Curran Associates, 2023.

[127] C. Liu and B. Wu. Evaluating Large Language Models on Graphs: Performance Insights and Comparative Analysis, 2023. arXiv:2308.11224.

[128] J. Liu, L. Cui, H. Liu, D. Huang, Y. Wang, and Y. Zhang. LogiQA: A Challenge Dataset for Machine Reading Comprehension with Logical Reasoning. In C. Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI '20, pages 3622–3628. International Joint Conferences on Artificial Intelligence Organization, Jul 2020.

[129] J. Liu, C. Yang, Z. Lu, J. Chen, Y. Li, M. Zhang, T. Bai, Y. Fang, L. Sun, P. S. Yu, and C. Shi. Towards Graph Foundation Models: A Survey and Beyond, 2023. arXiv:2310.11829.

[130] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. Pre-Train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. *ACM Comput. Surv.*, 55(9), Jan 2023.

[131] T. Liu, Q. Guo, Y. Yang, X. Hu, Y. Zhang, X. Qiu, and Z. Zhang. Plan, Verify and Switch: Integrated Reasoning with Diverse X-of-Thoughts. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, EMNLP '23, pages 2807–2822, Singapore, Dec 2023. Association for Computational Linguistics.

[132] Z. Liu, Z. Lai, Z. Gao, E. Cui, X. Zhu, L. Lu, Q. Chen, Y. Qiao, J. Dai, and W. Wang. ControlLLM: Augment Language Models with Tools by Searching on Graphs, 2023. arXiv:2310.17796.

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

[134] R. Long, P. Pasupat, and P. Liang. Simpler Context-Dependent Logical Forms via Model Projections. In K. Erk and N. A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL '16, pages 1456–1465, Berlin, Germany, Aug 2016. Association for Computational Linguistics.

[135] P. Lu, S. Mishra, T. Xia, L. Qiu, K.-W. Chang, S.-C. Zhu, O. Tafjord, P. Clark, and A. Kalyan. Learn to Explain: Multimodal Reasoning via Thought Chains for Science Question Answering. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems (NeurIPS '22)*, volume 35, pages 2507–2521. Curran Associates, 2022.

[136] P. Lu, B. Peng, H. Cheng, M. Galley, K.-W. Chang, Y. N. Wu, S.-C. Zhu, and J. Gao. Chameleon: Plug-and-Play Compositional Reasoning with Large Language Models. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems (NeurIPS '23)*, volume 36, pages 43447–43478. Curran Associates, 2023.

[137] P. Lu, L. Qiu, K.-W. Chang, Y. N. Wu, S.-C. Zhu, T. Rajpurohit, P. Clark, and A. Kalyan. Dynamic Prompt Learning via Policy Gradient for Semi-structured Mathematical Reasoning. In *Proceedings of the Eleventh International Conference on Learning Representations*, ICLR '23, May 2023.

[138] L. Luo, Y.-F. Li, G. Haffari, and S. Pan. Reasoning on Graphs: Faithful and Interpretable Large Language Model Reasoning, 2023. arXiv:2310.01061.

[139] A. Ma, X. Wang, J. Li, C. Wang, T. Xiao, Y. Liu, H. Cheng, J. Wang, Y. Li, Y. Chang, J. Li, D. Wang, Y. Jiang, L. Su, G. Xin, S. Gu, Z. Li, B. Liu, D. Xu, and Q. Ma. Single-cell biological network inference using a heterogeneous graph transformer. *Nature Communications*, 14(1), 2023.

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

[141] A. Madaan and A. Yazdanbakhsh. Text and Patterns: For Effective Chain of Thought, It Takes Two to Tango, 2022. arXiv:2209.07686.

[142] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky. The Stanford CoreNLP Natural Language Processing Toolkit. In K. Bontcheva and J. Zhu, editors, *Proceedings of 52nd Annual Meeting of the Association for Computational Lin-

guistics: System Demonstrations*, ACL '14, pages 55–60, Baltimore, Maryland, Jun 2014. Association for Computational Linguistics.

[143] R. Mirzaee and P. Kordjamshidi. Transfer Learning with Synthetic Corpora for Spatial Role Labeling and Reasoning. In Y. Goldberg, Z. Kozareva, and Y. Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, EMNLP '22, pages 6148–6165, Abu Dhabi, United Arab Emirates, Dec 2022. Association for Computational Linguistics.

[144] S. Mittal. A survey of FPGA-based accelerators for convolutional neural networks. *Neural Computing and Applications*, 32(4):1109–1139, Feb 2020.

[145] S. Mo and M. Xin. Tree of Uncertain Thoughts Reasoning for Large Language Models, 2023. arXiv:2309.07694.

[146] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun. A Modern Primer on Processing in Memory. In M. M. S. Aly and A. Chattopadhyay, editors, *Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann*, Computer Architecture and Design Methodologies (CADM), pages 171–243. Springer Nature Singapore, 2023.

[147] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun. Processing Data Where It Makes Sense: Enabling In-Memory Computation. *Microprocessors and Microsystems*, 67:28–41, Jun 2019.

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

[149] S. Pan, L. Luo, Y. Wang, C. Chen, J. Wang, and X. Wu. Unifying Large Language Models and Knowledge Graphs: A Roadmap, 2023. arXiv:2306.08302.

[150] S. Pan, Y. Zheng, and Y. Liu. Integrating Graphs with Large Language Models: Methods and Prospects, 2023. arXiv:2310.05499.

[151] A. Patel, S. Bhattamishra, and N. Goyal. Are NLP Models really able to Solve Simple Math Word Problems? In K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, and Y. Zhou, editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL '21, pages 2080–2094. Association for Computational Linguistics, Jun 2021.

[152] O. Press, M. Zhang, S. Min, L. Schmidt, N. Smith, and M. Lewis. Measuring and Narrowing the Compositionality Gap in Language Models. In H. Bouamor, J. Pino, and K. Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711, Singapore, Dec 2023. Association for Computational Linguistics.

[153] R. Puri, D. S. Kung, G. Janssen, W. Zhang, G. Domeniconi, V. Zolotov, J. Dolby, J. Chen, M. Choudhury, L. Decker, V. Thost, L. Buratti, S. Pujar, S. Ramji, U. Finkler, S. Malaika, and F. Reiss. CodeNet: A Large-Scale AI for Code Dataset for Learning a Diversity of Coding Tasks, 2021. arXiv:2105.12655.

[154] J. Qi, Z. Xu, Y. Shen, M. Liu, D. Jin, Q. Wang, and L. Huang. The Art OF SOCRATIC QUESTIONING: Recursive Thinking with Large Language Models, 2023. arXiv:2305.14999.

[155] C. Qian, H. Tang, Z. Yang, H. Liang, and Y. Liu. Can Large Language Models Empower Molecular Property Prediction?, 2023. arXiv:2307.07443.

[156] S. Qiao, Y. Ou, N. Zhang, X. Chen, Y. Yao, S. Deng, C. Tan, F. Huang, and H. Chen. Reasoning with Language Model Prompting: A Survey. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL '23, pages 5368–5393, Toronto, Canada, Jul 2023. Association for Computational Linguistics.

[157] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language Models are Unsupervised Multitask Learners. https://openai.com/research/better-language-models, 2019. (accessed: Dec. 4, 2023).

[158] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer, 2023. arXiv:1910.10683.

[159] A. Rasouli, I. Kotseruba, T. Kunic, and J. Tsotsos. PIE: A Large-Scale Dataset and Models for Pedestrian Intention Estimation and Trajectory Prediction. In *Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision*, ICCV '19, pages 6261–6270, 2019.

[160] T. B. Richards. AutoGPT: build & use AI agents - Github. https://github.com/Significant-Gravitas/AutoGPT, Mar 2023. (accessed Jan. 23, 2024).

[161] S. Roy and D. Roth. Solving General Arithmetic Word Problems. In L. Màrquez, C. Callison-Burch, and J. Su, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, EMNLP '15, pages 1743–1752, Lisbon, Portugal, Sep 2015. Association for Computational Linguistics.

[162] S. Saha, O. Levy, A. Celikyilmaz, M. Bansal, J. Weston, and X. Li. Branch-Solve-Merge Improves Large Language Model Evaluation and Generation, 2023. arXiv:2310.15123.

[163] M. Sakarvadia, A. Ajith, A. Khan, D. Grzenda, N. Hudson, A. Bauer, K. Chard, and I. Foster. Memory Injections: Correcting Multi-Hop Reasoning Failures during Inference in Transformer-Based Language Models, 2023. arXiv:2309.05605.

[164] A. Saparov and H. He. Language Models Are Greedy Reasoners: A Systematic Formal Analysis of Chain-of-Thought. In *Proceedings of the Eleventh International Conference on Learning Representations*, ICLR '23, May 2023.

[165] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language Models Can Teach Themselves to Use Tools, 2023. arXiv:2302.04761.

[166] B. Sel, A. Al-Tawaha, V. Khattar, L. Wang, R. Jia, and M. Jin. Algorithm of Thoughts: Enhancing Exploration of Ideas in Large Language Models, 2023. arXiv:2308.10379.

[167] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry. Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '17, pages 273–287, Cambridge, Massachusetts, 2017. Association for Computing Machinery.

[168] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems (NeurIPS '23)*, volume 36, pages 8634–8652. Curran Associates, 2023.

[169] M. Shridhar, X. Yuan, M.-A. Côté, Y. Bisk, A. Trischler, and M. Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *Proceedings of the International Conference on Learning Representations*, ICLR '21, 2021.

[170] C. Singh, J. Morris, A. Rush, J. Gao, and Y. Deng. Tree Prompting: Efficient Task Adaptation without Fine-Tuning. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, EMNLP '23, pages 6253–6267, Singapore, Dec 2023. Association for Computational Linguistics.

[171] C. Singh, J. X. Morris, J. Aneja, A. M. Rush, and J. Gao. Explaining Patterns in Data with Language Models via Interpretable Autoprompting, 2022. arXiv:2210.01848.

[172] A. Srivastava et al. Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023.

[173] I. Stelmakh, Y. Luan, B. Dhingra, and M.-W. Chang. ASQA: Factoid Questions Meet Long-Form Answers, 2022. arXiv:2204.06092.

[174] A. Strausz, F. Vella, S. Di Girolamo, M. Besta, and T. Hoefler. Asynchronous Distributed-Memory Triangle Counting and LCC with RMA Caching. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, IPDPS '22, pages 291–301, Lyon, France, Jun 2022.

[175] A. Suhr, M. Lewis, J. Yeh, and Y. Artzi. A Corpus of Natural Language for Visual Reasoning. In R. Barzilay and M.-Y. Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, ACL '17, pages 217–223, Vancouver, Canada, Jul 2017. Association for Computational Linguistics.

[176] J. Sun, C. Xu, L. Tang, S. Wang, C. Lin, Y. Gong, H.-Y. Shum, and J. Guo. Think-on-Graph: Deep and Responsible Reasoning of Large Language Model on Knowledge Graph, 2023. arXiv:2307.07697.

[177] X. Sun, J. Zhang, X. Wu, H. Cheng, Y. Xiong, and J. Li. Graph Prompt Learning: A Comprehensive Survey and Beyond, 2023. arXiv:2311.16534.

[178] Y. Sun and J. Han. *Mining Heterogeneous Information Networks: Principles and Methodologies*. Synthesis Lectures on Data Mining and Knowledge Discovery (SLDMKD). Springer International Publishing, 2012.

[179] O. Tafjord, B. Dalvi, and P. Clark. ProofWriter: Generating Implications, Proofs, and Abductive Statements over Natural Language. In C. Zong, F. Xia, W. Li, and R. Navigli, editors, *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3621–3634. Association for Computational Linguistics, Aug 2021.

[180] A. Talmor, J. Herzig, N. Lourie, and J. Berant. CommonsenseQA: A Question Answering Challenge Targeting Commonsense Knowledge. In J. Burstein, C. Doran, and T. Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, NAACL '19, pages 4149–4158, Minneapolis, Minnesota, Jun 2019. Association for Computational Linguistics.

[181] J. Tang, Y. Yang, W. Wei, L. Shi, L. Su, S. Cheng, D. Yin, and C. Huang. GraphGPT: Graph Instruction Tuning for Large Language Models, 2023. arXiv:2310.13023.

[182] A. J. Thirunavukarasu, D. S. J. Ting, K. Elangovan, L. Gutierrez, T. F. Tan, and D. S. W. Ting. Large language models in medicine. *Nature Medicine*, 29(8):1930–1940, Jul 2023.

[183] Y. Tian, H. Song, Z. Wang, H. Wang, Z. Hu, F. Wang, N. V. Chawla, and P. Xu. Graph Neural Prompting with Large Language Models, 2023. arXiv:2309.15427.

[184] H. Trivedi, N. Balasubramanian, T. Khot, and A. Sabharwal. MuSiQue: Multihop Questions via Single-hop Question Composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

[185] R. Tutunov, A. Grosnit, J. Ziomek, J. Wang, and H. Bou-Ammar. Why Can Large Language Models Generate Correct Chain-of-Thoughts?, 2023. arXiv:2310.13571.

[186] H. Wang, S. Feng, T. He, Z. Tan, X. Han, and Y. Tsvetkov. Can Language Models Solve Graph Problems in Natural Language? In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems (NeurIPS '23)*, volume 36, pages 30840–30861. Curran Associates, 2023.

[187] H. Wang, Y. Gao, X. Zheng, P. Zhang, H. Chen, and J. Bu. Graph Neural Architecture Search with GPT-4, 2023. arXiv:2310.01436.

[188] L. Wang, W. Xu, Y. Lan, Z. Hu, Y. Lan, R. K.-W. Lee, and E.-P. Lim. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. In A. Rogers, J. Boyd-Graber, and N. Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL '23, pages 2609–2634, Toronto, Canada, Jul 2023. Association for Computational Linguistics.

[189] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks, 2020. arXiv:1909.01315.

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

[191] X. Wang, Q. Yang, Y. Qiu, J. Liang, Q. He, Z. Gu, Y. Xiao, and W. Wang. KnowledGPT: Enhancing Large Language Models with Retrieval and Storage Access on Knowledge Bases, 2023. arXiv:2308.11761.

[192] Y. Wang, N. Lipka, R. A. Rossi, A. Siu, R. Zhang, and T. Derr. Knowledge Graph Prompting for Multi-Document Question Answering, 2023. arXiv:2308.11730.

[193] Z. Wang, S. Cai, G. Chen, A. Liu, X. S. Ma, and Y. Liang. Describe, Explain, Plan and Select: Interactive Planning with LLMs Enables Open-World Multi-Task Agents. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems (NeurIPS '23)*, volume 36, pages 34153–34189. Curran Associates, 2023.

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

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

[196] X. Wei, S. Wang, D. Zhang, P. Bhatia, and A. Arnold. Knowledge Enhanced Pretrained Language Models: A Comprehensive Survey, 2021. arXiv:2110.08455.

[197] Y. Wen, Z. Wang, and J. Sun. MindMap: Knowledge Graph Prompting Sparks Graph of Thoughts in Large Language Models, 2023. arXiv:2308.09729.

[198] Y. Weng, M. Zhu, F. Xia, B. Li, S. He, S. Liu, B. Sun, K. Liu, and J. Zhao. Large Language Models are Better Reasoners with Self-Verification. In H. Bouamor, J. Pino, and K. Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2550–2575, Singapore, Dec 2023. Association for Computational Linguistics.

[199] J. Weston and S. Sukhbaatar. System 2 Attention (is something you might need too), 2023. arXiv:2311.11829.

[200] N. Wies, Y. Levine, and A. Shashua. The Learnability of In-Context Learning, 2023. arXiv:2303.07895.

[201] T. Wu, E. Jiang, A. Donsbach, J. Gray, A. Molina, M. Terry, and C. J. Cai. PromptChainer: Chaining Large Language Model Prompts through Visual Programming. In *Extended Abstracts of the Conference on Human Factors in Computing Systems*, CHI EA '22, New Orleans, LA, USA, May 2022. Association for Computing Machinery.

[202] X. Wu, K. Zhou, M. Sun, X. Wang, and N. Liu. A Survey of Graph Prompting Methods: Techniques, Applications, and Challenges, 2023. arXiv:2303.07275.

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

[204] S. M. Xie, A. Raghunathan, P. Liang, and T. Ma. An Explanation of In-context Learning as Implicit Bayesian Inference. In *Proceedings of the Tenth International Conference on Learning Representations*, ICLR '22, Apr 2022.

[205] Y. Xie, K. Kawaguchi, Y. Zhao, J. X. Zhao, M.-Y. Kan, J. He, and M. Xie. Self-Evaluation Guided Beam Search for Reasoning. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems (NeurIPS '23)*, volume 36, pages 41618–41650. Curran Associates, 2023.

[206] B. Xu, Q. Wang, Z. Mao, Y. Lyu, Q. She, and Y. Zhang. *k*NN Prompting: Beyond-Context Learning with Calibration-Free Nearest Neighbor Inference. In *Proceedings of the Eleventh International Conference on Learning Representations*, ICLR '23, May 2023.

[207] C. Xu, Q. Sun, K. Zheng, X. Geng, P. Zhao, J. Feng, C. Tao, and D. Jiang. WizardLM: Empowering Large Language Models to Follow Complex Instructions, 2023. arXiv:2304.12244.

[208] C. Yang, D. Bo, J. Liu, Y. Peng, B. Chen, H. Dai, A. Sun, Y. Yu, Y. Xiao, Q. Zhang, C. Wang, Y. Guo, and C. Shi. Data-centric Graph Learning: A Survey, 2023. arXiv:2310.04987.

[209] J. Yang, G. Xiao, Y. Shen, W. Jiang, X. Hu, Y. Zhang, and J. Peng. A Survey of Knowledge Enhanced Pre-trained Models, 2021. arXiv:2110.00269.

[210] L. Yang, H. Chen, Z. Li, X. Ding, and X. Wu. ChatGPT is not Enough: Enhancing Large Language Models with Knowledge Graphs for Fact-aware Language Modeling, 2023. arXiv:2306.11489.

[211] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. Cohen, R. Salakhutdinov, and C. D. Manning. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, EMNLP '18, pages 2369–2380, Brussels, Belgium, Nov 2018. Association for Computational Linguistics.

[212] F. Yao, C. Tian, J. Liu, Z. Zhang, Q. Liu, L. Jin, S. Li, X. Li, and X. Sun. Thinking Like an Expert: Multimodal Hypergraph-of-Thought (HoT) Reasoning to boost Foundation Modals, 2023. arXiv:2308.06207.

[213] S. Yao, D. Yu, J. Zhao, I. Shafran, T. Griffiths, Y. Cao, and K. Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems (NeurIPS '23)*, volume 36, pages 11809–11822. Curran Associates, 2023.

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

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

[216] R. Ye, C. Zhang, R. Wang, S. Xu, and Y. Zhang. Natural Language is All a Graph Needs, 2023. arXiv:2308.07134.

[217] Y. Ye, B. Hui, M. Yang, B. Li, F. Huang, and Y. Li. Large Language Models Are Versatile Decomposers: Decomposing Evidence and Questions for Table-Based Reasoning. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, pages 174–184, Taipei, Taiwan, 2023.

[218] J. Yu, R. He, and R. Ying. Thought Propagation: An Analogical Approach to Complex Reasoning with Large Language Models, 2023. arXiv:2310.03965.

[219] L. Zeit-Altpeter. Forward Looking Active Retrieval Augmented Generation. In *Proceedings of the AI Summer School at Uni Jena (Poster)*, 2023.

[220] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla. Heterogeneous Graph Neural Network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 793–803, Anchorage, AK, USA, 2019. Association for Computing Machinery.

[221] H. Zhang, L. H. Li, T. Meng, K.-W. Chang, and G. V. den Broeck. On the Paradox of Learning to Reason from Data, 2022. arXiv:2205.11502.

[222] J. Zhang. Graph-ToolFormer: To Empower LLMs with Graph Reasoning Ability via Prompt Augmented by ChatGPT, 2023. arXiv:2304.11116.

[223] Y. Zhang, Z. Chen, W. Zhang, and H. Chen. Making Large Language Models Perform Better in Knowledge Graph Completion, 2023. arXiv:2310.06671.

[224] Y. Zhang, J. Yang, Y. Yuan, and A. C.-C. Yao. Cumulative Reasoning with Large Language Models, 2023. arXiv:2308.04371.

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

[226] Z. Zhang, H. Li, Z. Zhang, Y. Qin, X. Wang, and W. Zhu. Graph Meets LLMs: Towards Large Graph Models, 2023. arXiv:2308.14522.

[227] Z. Zhang, X. Wang, Z. Zhang, H. Li, Y. Qin, S. Wu, and W. Zhu. LLM4DyG: Can Large Language Models Solve Problems on Dynamic Graphs?, 2023. arXiv:2310.17110.

[228] Z. Zhang, Y. Yao, A. Zhang, X. Tang, X. Ma, Z. He, Y. Wang, M. Gerstein, R. Wang, G. Liu, and H. Zhao. Igniting Language Intelligence: The Hitchhiker's Guide From Chain-of-Thought Reasoning to Language Agents, 2023. arXiv:2311.11797.

[229] J. Zhao, L. Zhuo, Y. Shen, M. Qu, K. Liu, M. Bronstein, Z. Zhu, and J. Tang. GraphText: Graph Reasoning in Text Space, 2023. arXiv:2310.01089.

[230] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen. A Survey of Large Language Models, 2023. arXiv:2303.18223.

[231] X. Zhao, M. Li, W. Lu, C. Weber, J. H. Lee, K. Chu, and S. Wermter. Enhancing Zero-Shot Chain-of-Thought Reasoning in Large Language Models through Logic, 2023. arXiv:2309.13339.

[232] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems (NeurIPS '23)*, volume 36, pages 46595–46623. Curran Associates, 2023.

[233] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. V. Le, and E. H. Chi. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. In *Proceedings of the Eleventh International Conference on Learning Representations*, ICLR '23, May 2023.

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

[235] Y. Zhu, X. Wang, J. Chen, S. Qiao, Y. Ou, Y. Yao, S. Deng, H. Chen, and N. Zhang. LLMs for Knowledge Graph Construction and Reasoning: Recent Capabilities and Future Opportunities, 2023. arXiv:2305.13168.

[236] Y. Zong, O. Mac Aodha, and T. Hospedales. Self-Supervised Multimodal Learning: A Survey, 2023. arXiv:2304.01008.

APPENDIX A
DETAILED DESCRIPTIONS OF CHAIN SCHEMES

We study prompting schemes that use chain topologies. We list them and determine their relationship to our blueprint in Table 1.

A.1 Single-Prompt Schemes

Chain-based reasoning can be conducted within a single prompt, potentially containing multiple in-context examples. It is used in both textual and visual reasoning tasks.

In the seminal **Chain-of-Thought (CoT)** [195] scheme, in-context examples consisting of intermediate reasoning steps for a given complex question are introduced to prompt the LLM to generate a similar multi-step reasoning chain. This chain is used to improve the quality of the LLM answer compared to basic IO prompting.

Zero-shot-CoT was proposed by Kojima et al. [112]. It comes with multi-step reasoning *without* in-context examples, by simply prompting the LLM with one sentence, *"Let's think step by step"*, or using other similar statements. This zero-shot prompting improves upon other zero-shot methods in terms of the accuracy of reasoning outcomes.

SelfAsk [152] is a single-prompt scheme similar to CoT which augments the in-context examples with intermediate questions. Instead of only providing a step-by-step reasoning chain in the examples, each step in this chain is expanded to pose a follow-up question and then answer it, e.g., *"Superconductivity was discovered in 1911."* is augmented to *"Follow up: When was superconductivity discovered? Intermediate answer: Superconductivity was discovered in 1911"*.

Plan-and-Solve (PS) Prompting [188] builds a chain based on a devised plan, harnessing Zero-shot CoT. It first divides the complex task into a list of sub-tasks and then executes the stepwise plan for the solution. PS can be extended to **PS+** by adding instructions in the prompt to request the LLM to extract variables and explicitly calculate the intermediate values. PS+ has no impact on the length of the chain.

Program of Thoughts (PoT) [41] is a single-prompt scheme similar to CoT which generates code to solve a question. Rather than the natural-language-based examples used in CoT, the LLM is prompted by code exemplars, where each statement refers to one intermediate step in the reasoning chain. Zero-shot prompting can also be applied, e.g., *"Let's write a Python program step by step and return the result. Firstly we need to define the variables."*. The final result is obtained by executing the generated code.

A.2 Multi-Prompt Schemes

Elicited by multiple rounds of prompting, the LLM reasoning process can be a chain consisting of several messages.

Selection-Inference (SI) [51] aims to address multi-step logical reasoning problems where all necessary information is provided within the input context. Each reasoning step in SI involves two distinct sub-tasks: selection and inference. Firstly, the selection sub-task is responsible for identifying and selecting relevant information needed for the subsequent reasoning step, effectively pruning the context. Following this, the inference sub-task generates a new intermediate piece of information, thereby performing the actual reasoning step. This newly generated information then becomes available for use in subsequent reasoning steps for the selection sub-task. The length of the chain is bounded to a fixed number of steps, and the prompts for both sub-tasks include relevant few-shot examples.

Chain-of-symbol (CoS) [89] is a multi-prompt scheme utilizing two prompts for solving spatial planning tasks in natural language. Prompted in a zero-shot manner, the LLM initially generates CoT-like in-context exemplars and modifies them by replacing the spatial relationships with symbols. For example, the navigational task *"Start at bank A. There are two stores on the map, store B and store G. The road from bank A to store B goes through bank C, house H, and cinema F to store B, totaling 600 meters."* is succinctly converted into the symbol sequence *"bank A / bank C / house H / cinema F / store B, (200 + 100 + 100 + 200 = 600)."*. Exemplars simplified in this way are then provided as prompts for LLMs to conduct CoT-based reasoning.

A.2.1 Multi-Prompt Chains with Decomposition

The introduction of a node for decomposition at the start of a reasoning chain can not only determine the chain's depth, but also facilitate more detailed reasoning approaches in the subsequent sub-steps. This allows for a finer-grained resolution of sub-questions, enriching the overall reasoning process. Apart from the schemes described below, similar strategies are applied in works [59], [102], [217].

Least-to-Most Prompting [233] grows a reasoning chain, where the decomposition of complex tasks is conducted in the first node and the sub-tasks are solved in the subsequent nodes. Prompted with few-shot examples showcasing the decomposition of questions into lists of sub-questions, the LLM firstly generates a sequence of sub-questions given an original question. Then the solving process starts by providing the LLM with example solutions for simple questions. Subsequently during each sub-step, the LLM generates the sub-solution, given the question for this step and the list of previously solved questions with the generated answers. The chain terminates when the final answer is returned after solving all sub-problems in the list.

Decomposed Prompting [105] is a modular framework for a detailed decomposition of complex tasks. To generate a reasoning chain, the LLM is prompted by demonstrations comprised of sequential question-operation triplets, which form "sub-questions". The growth of the chain follows the autoregressive pattern of CoT, while each sub-step captures the input and passes the output through the contextual environment. Under the limited scope of the tasks, the types of operation functions are fixed, enabling the LLM to conduct in-context learning for operations before inference.

A.2.2 Multi-Prompt Chains with Verification

The introduction of verification enables the reasoning frameworks to iteratively refine the generated context. With this strategy, the execution of chain-based reasoning is effectively extended with loops, with conditions on how many times one can loop over a node (based on the number

of iterations or some terminal conditions). Apart from the schemes described below, similar strategies are applied in works [123], [126], [198].

LogiCoT [231] is a zero-shot, multi-prompt framework that leverages iterative verification to minimize cumulative errors in the reasoning chain. After generating a reasoning chain through few-shot CoT, this framework systematically navigates the reasoning chain with zero-shot prompts to pinpoint the first instance where the original reasoning deviated. It preserves the accurate part of the reasoning chain up to the erroneous node and then prompts the LLM to diagnose and correct this specific misstep, thereby forming an alternative reasoning chain. This process repeats until a fully verified reasoning chain is established, with each node passing the verification.

SELF-REFINE [140] adopts a distinct approach to iterative refinement compared to LogiCoT. Instead of focusing on pinpointing and correcting errors within a reasoning chain, it initially generates a complete output and then enters a cyclical process, alternating between generating specific feedback with concrete actions for enhancement and applying this feedback to refine the output.

Similar to SELF-REFINE, **Reflexion** [168] employs the concept of iterative refinement through generating and applying feedback, but introduces a modular approach to the process, segmenting the refinement into specialized stages. It starts with an initial output by the LLM, followed by an evaluation through task-specific grading functions, and then generates informative feedback, which is subsequently applied. This reasoning chain continues looping until the evaluation stage confirms that the output is correct.

Reasoning Graph Verifier (RGV) [35] is a framework that addresses arithmetic questions. Their approach is a variant of CoT-SC, in which they employ a trainable verifier module to decide on the best CoT solution.

A.2.3 Multi-Prompt Chains with External Tools

To better integrate multiple execution methods, some schemes opt to devise a plan that specifies tools for handling each sub-task, before executing the reasoning chain.

Plan, Verify and Switch (PVS) [131] iteratively engages in planning, verification, and switching among different methods until the acquisition of the final answer. Initially, the LLM is prompted with definitions of three chain-based reasoning methods, which include CoT, PoT, and a method using the solution of linear equations to represent the reasoning process (named EoT). Then, at the first stage of each iteration, the LLM selects a reasoning method according to the input question and subsequently follows in-context examples on the application of the method to complete a reasoning chain accordingly. Then, the verification module acquires conditional information and the intermediate variables from the sub-steps of the chain and executes calculations, validating the correctness of the final answer. If the verification fails, the next iteration is activated, in which the LLM selects a new method from the non-selected ones and generates a new chain without memories of the previous reasoning process. The planning module facilitates the dynamic switch of reasoning methods, thereby integrating the strengths of different paradigms.

Chameleon [136] is a reasoning framework that provides the LLM with access to various external tools, such as programming interpreters and table readers. Given the question, tool descriptions, and tool usage examples, the LLM generates a plan for the reasoning chain, in which each node corresponds to a tool used in the sub-step. During the execution of the chain, the output of previous steps together with the cached tool execution results are taken as the input of each current step. The nodes of the chain are executed sequentially until the final result is returned.

There are also tool-based chain schemes with no planning module; they dynamically select the reasoning method.

ChatCoT [45] realizes tool-based reasoning chains through multi-round conversations with the LLM. Initially, the LLM is shown how to decompose CoT reasoning as a multi-turn conversation, in order to learn problem-solving in a step-by-step, autoregressive conversational way. Additionally, the knowledge of tools and relevant examples are included in the context to elicit their selection and correct usage. During inference, the chain proceeds and unfolds without a premature plan until the problem is solved, while the LLM simultaneously selects and executes appropriate tools during the multi-turn conversation. The conversation is constrained to a maximum number of turns, thereby constraining the maximum reasoning depth, while ensuring all previous turns are kept within the context window.

APPENDIX B
DETAILED DESCRIPTIONS OF TREE SCHEMES

We next analyze works that harness trees as the prompting topology. We structure the discussion based on the harnessed topology variants, see Figure 9.

B.1 Trees of Chains

While harnessing trees as prompting topologies has been explicitly established in the works by Long [133] and Yao [213], this idea has been present earlier. Specifically, **Chain-of-Thought with Self-Consistency (CoT-SC)** [190] is an early scheme that harnesses the tree structure to a certain degree. Here, multiple CoTs originate from the same initial (root) prompt, forming a "tree of chains". The chain providing the best outcome to the initial question, is selected as the final answer.

B.2 Single-Level Trees

Skeleton-of-Thought (SoT) [148] is a prompting scheme aiming to reduce the end-to-end generation latency of LLMs, caused by their inherent sequential decoding. Instead of generating one long continuous answer, this scheme uses a divide-and-conquer approach. In a first prompt, the LLM is instructed to generate a skeleton of the answer, i.e., a list of points that are independently answerable. Then, for each of these points, a new prompt is issued in parallel to answer just this specific part of the question. As these points are processed in parallel, the overall latency is reduced. One can additionally include a prompt at the beginning which lets the LLM decide on solving the question via independent points, or – whenever reasonable – by using a single continuous answer.

B.3 k–Ary Trees

First, the **Tree-of-Thought (ToT) design by Long** [133] utilizes a tree structure to decompose a problem into sub-problems and solve them using separate LLM prompts. After the LLM suggests possible next steps and corresponding partial solutions, a *checker module* decides if any of these solutions is valid, whether it can be selected as the final one, or whether it should backtrack to the previous step. All issued prompts and answers are explicitly stored as a tree structure and navigated through using a *controller module*. The LLM prompting is only used to generate the next individual steps (i.e., hops) in this tree, whereas the overall problem solving process is coordinated by the controller.

Tree of Thoughts (ToT) by Yao et al. [213] differs from the above ToT approach in using the LLM itself as a solution *evaluator* with access to all generated solutions, instead of using a programmed or learned evaluator module. This allows to rate states individually or vote across intermediate solutions to select the most promising one to continue with the search. Both mentioned ToT approaches are a generalization of the IO, CoT, and CoT-SC prompting schemes.

B.3.1 Pre-ToT Schemes

Thought Decomposition [205] is a multi-prompt scheme based on stochastic beam search and self-evaluation. At each level (reasoning step), n new intermediate nodes are generated for each of the (usually) k input nodes. Each node of this set of nk nodes is evaluated and then the set is pruned down to k output nodes via sampling with stochastic beam search. For both the generation and the evaluation prompts, few-shot examples are used with a focus on textual or program-aided reasoning, depending on the dataset used for evaluation.

Creswell and Shanahan [50] describe an extension of the chain-based Selection-Inference [51], which is used to answer multiple-choice questions by chaining together correct reasoning steps over a predefined context, i.e., a set of statements (the provided statements are sufficient to derive the answer to the question). The space of all possible reasoning chains forms a tree rooted at the initial question. Each node represents a reasoning step that is derived by (1) selecting a subset of statements from the context and then (2) inferring a new statement, which is then added to the context. For both steps, the authors use separately fine-tuned LMs. This tree is explored using beam search, where another fine-tuned LM assesses the value of the current node, and the search ends as soon as a halter-LM decides that the question can be answered with the current context and then answers it.

Dynamic Least-to-Most Prompting [58] extends least-to-most prompting with a tree-based problem decomposition and a dynamic external tree-based few-shot example selection. The goal is to turn natural language questions into formal representations such as SPARQL queries. Initially the input question is decomposed into sub-problems using a series of prompts. This process yields an arbitrary decision tree for the input in contrast to a chain in traditional least-to-most prompting. This decision tree is used to dynamically select few-shot examples by externally matching that tree against a precomputed decision tree for examples. In the final step, the decision tree is linearized into a sequence of sub-problems with increasing complexity which are then translated by using prompts enhanced with the respective examples selected in the second step.

B.3.2 Post-ToT Schemes

Different schemes have extended the initial ToT designs. **Algorithm of Thoughts (AoT)** [166] is a single-prompt approach that utilizes in-context examples formulated in an algorithmic fashion. Instead of providing step-by-step examples as in CoT prompting, AoT harnesses algorithmic reasoning steps in the examples, that explore the solution space (tree) with either DFS or BFS. As opposed to ToT, AoT uses only a single prompt.

Tree of Uncertain Thought (TouT) [145] extends ToT with local "uncertainty scores" by incorporating the variance of multiple LLM responses into the state evaluation function.

Tree-of-Mixed-Thought (TomT) [91] uses ToT-based prompting to answer questions on visual scene graphs, e.g., *"Does the red chair have the same number of legs as the table in front of it?"*. The LLM is tasked to generate Python code that solves the question using a set of provided functions to extract and reason on the data of the scene graph. This prompting scheme uses a DFS variant of ToT where each individual node generates up to s consecutive reasoning steps, i.e. lines of code, opposed to the original ToT where $s = 1$. Evaluating the correctness of the reasoning steps is done as in ToT for each node, where one checks if the generated code so far can (1) run correctly, and (2) the interactions with the scene graph are consistent with the existing elements in the scene.

Tree of Clarifications (ToC) [106] enables to answer ambiguous questions by first retrieving pertinent external information and then recursively prompting an LLM to construct a tree of disambiguations for the initial question. This tree is explored using BFS and after no more ambiguations exist, a long form answer is generated, by combining all the previously explored nodes.

Tree Prompting [170] is a high-level approach that proposes the training of a binary decision tree to classify text during inference. Several mechanisms to construct the prompts that make up the nodes of the decision tree during learning are proposed: prompts based on few-shot examples, human curated instruction prompts, dynamic prompts constructed with discrete prompt search methods like iPrompt [171], and kNN prompting [206], which constructs new nodes based on the nearest neighbors. kNN prompting can result in multiple prompts per decision tree node with the other mechanisms using only a single prompt. Additional improvements can be made by employing tree ensembles such as random forests [120] or gradient-boosted trees [64]. During inference the number of prompts is bound by the depth of the decision tree, where as the training can be resource intensive.

B.4 Analysis & Comparison of Designs

We now broadly discuss and analyze tree designs with respect to different aspects of our blueprint. A detailed analysis can be found in Appendix E.2.

B.4.1 Topology & Its Construction

The key novel architectural feature of tree schemes is **exploration** of a step, i.e., the ability to generate multiple new steps based on a given single one. The vast majority of tree schemes are multi-prompt. Most multi-prompt schemes use a dynamic approach to building the tree topology. The details of how the topology is exactly shaped depend on the specific question. For most multi-prompt approaches, the user can adapt the tree topology to a certain degree, i.e., by varying the branching factor and limiting the depth of the tree.

B.4.2 Performance

Increasing the **branching factor** often leads to more **diversity** of outcomes, which can be beneficial for accuracy, but also increases the number of prompts, i.e., computational cost. The most advantageous branching factor is hard to find and often depends on the specific problem to solve. Easily decomposable problems may benefit less from more branching than complex problems. Specifically, more complicated problems profit more from decomposing them into many/diverse sub-problems (e.g., this ensures enough diversity for self-consistency to work better). In contrast, a question that has clearly only two sub-parts does not benefit from many more subdivisions, as the additional branches can either be redundant or wrong. Single-prompt approaches can perform better on some problems than multi-prompt approaches, while using only a single prompt compared to possibly hundreds.

APPENDIX C
DETAILED DESCRIPTIONS OF GRAPH SCHEMES

We finally describe schemes with topologies beyond trees or chains. We observe that they can be further grouped into subclasses, based on the harnessed class of graphs. We picture different types of graph-based schemes in Figure 9.

C.1 Special Classes of Graphs

Different schemes harness certain classes of graphs.

Branch-Solve-Merge (BSM) [162] employs a 1–level double tree structure to first divide a problem into independently solvable sub-problems, and then combine them into a final solution. The first prompt instructs the LLM to propose sub-problems, which are then solved independently. The final prompt instructs the LLM to merge the results of the sub-problems into a single output.

Thought Propagation (TP) [218] employs a multi–level double tree structure for prompting the LLM. This approach follows the idea that multi-step problems are easier to solve with access to example solutions of similar problems. This is done in three steps. In the first step, given an input problem the LLM is prompted to propose a set of related problems. The main idea here is that solving these related problems can then be used as reference examples for solving the input problem instead of reasoning from scratch. Additionally, solving similar problems can lead to high-level plans for the input problem and allows TP to rectify errors during planning. The LLM is prompted using in-context examples to generate related problems for both situations.

In the second step, the LLM is prompted to solve the input problem as well as the related problems. For this prompt, existing prompting techniques, e.g., CoT, ToT, etc., can be used. Even though the solutions to the related problems are not expert-level, they can be refined in the next step. In the last step, the LLM is prompted to come up with new solutions for the initial problem based on the solutions from the related problems. Additionally, the LLM is prompted to derive high-level plans to solve the input problem using the solutions of the related problems. This process can also be extended to recursively generate further related problems, which yields in the general form a double tree structure with depth k.

Socratic Questioning [154] is a prompting scheme that models recursive exploration of the thought space using a tree structure. Hereby, the original question is recursively decomposed into sub-tasks until all tasks can be solved with high confidence. These results are then aggregated and propagated back up the tree to answer the original question. This results in an overall double tree reasoning topology. Additionally, the approach also provides a model to generate an image caption related to the text prompt and thus allows multi-modal reasoning.

C.2 Directed Graphs

Some schemes embrace a general directed graph model.

Graph of Thoughts (GoT) [10] uses a multi-prompt approach to improve the LLM problem solving performance by decomposing a given task into sub-tasks that form a graph. This decomposition is specified as a *Graph of Operations (GoO)*. The GoO coordinates how the LLM is prompted and how the results, which form a separate graph called *Graph Reasoning State (GRS)*, are further used in the reasoning process.

Graph of Thought [119] presents a multi-prompt approach where a graph is constructed recursively in a DFS manner by starting at the *question node* that represents the question to be answered by the LLM. From this node, possible reasoning paths are generated by the LLM. For each path, new nodes, i.e., intermediate reasoning steps, are generated by the LLM and are then used to grow the graph. To limit the size of the graph, the scheme uses a depth limit and requires a set of *condition nodes* to be provided at the start. These nodes represent axioms for the reasoning process that form initial nodes in the reasoning paths. After the graph construction, a path from the condition nodes to the question node is searched, and a checker module validates each reasoning step along this path. This checking is done by a series of calls to the LLM and only passes with a positive score, if all calls agree on the validity of the step. If no valid path is found, the graph is updated to enable valid reasoning paths. In the first step of the update, all nodes only depending on condition nodes (via a valid step) are added to the condition set. Then, new nodes and edges are added to the existing graph in the same DFS manner as when creating the graph from scratch. These graph updates are repeated until a valid path to the question node is found and therefore a solution to the initial question.

Graph-of-Thought [215] describes a two-stage framework to answer multi-modal questions, i.e., textual questions accompanied by images. In the first stage, the model

generates natural language rationales based on the input text, which provide additional context and knowledge to support answering the given question. This rationale generation is learned as part of the overall model pipeline. In the second stage, these rationales are then appended to the initial question and passed again through the model to predict an answer. The prediction consists of multiple steps. First, subject-verb-object triples are extracted from the input question using the Stanfords OpenIE system [3]. Using these triples, a Graph-of-Thought is constructed by first converting each triple to a 3–node path and then merging similar nodes using the Stanford CoreNLP system [142]. Next, the input text and the accompanying image are encoded using a Transformer encoder and a vision encoder, respectively. The Graph-of-Thought is encoded using a GAN. The resulting features from text and image are then combined using cross-attention and passed through a gated fusion layer before passing them to the final Transformer decoder that predicts the rationales in the first stage and the answers in the second stage. The scheme uses the pre-trained T5 [158] model and fine-tunes it for the rationale generation.

ControlLLM [132] is a framework that allows answering multi-modal queries, e.g. understanding and generating images, videos, and audio. First, the LLM is prompted to decompose the query into predefined sub-tasks, e.g., "image-processing" or "video-generation". Hereby, the LLM also infers appropriate input and output types, e.g., text or image, from a predefined set for each sub-task. In a next step, a so called *Tool Graph* is constructed by using the generated sub-tasks and the input resources, e.g. video or text, as nodes. Connections between the nodes are drawn for all compatible input/output types. Nodes corresponding to sub-tasks or resources are called *tool nodes* or *resource nodes* respectively. In a next step, the graph is searched in a DFS manner until a solution, i.e., path from input to output resource node, is found. The authors evaluated different heuristics for the DFS, i.e., greedy search, beam search, adaptive beam search and exhaustive search. The graph traversal is guided by the LLM and in each step the LLM is prompted to rate the relevance of the connected tool nodes for solving the overall task, this rating is then used as the heuristic. In a last step, the found solution, i.e., path from input to output in the tool graph, is executed to retrieve the requested output. Finally, the LLM is prompted to summarize the result for the user. The overall approach is similar to GoT [10], with the main difference being that each node can access and use a tool and the graph traversal uses a LLM-based heuristic.

Cumulative Reasoning [224] describes a paradigm to solve multi-step problems by iteratively constructing a directed acyclic graph (DAG). A proposer module suggest a next deduction step based on any previous steps, a verifier module evaluates the proposed step and finally, a reporter module checks if a valid solution has been reached and otherwise a next step is proposed. In this approach, one manually writes problem specific prompts that follow this paradigm. One also manually provides the implementation on how the individual modules interact, i.e., how the DAG is formed. Conceptually, this approach is an extension of ToT as it allows each step to use all previously derived results, meaning different sub trees can be connected to form a DAG.

Everything of Thoughts (XoT) [57] is a two-stage framework utilizing a reinforcement learning model trained using Monte Carlo Tree Search on specific problem sets to generate graphs of thoughts as well as iteratively prompting the LLM to revise and infer solutions based on these graphs. First, the model has to be trained in advance and can be applied to solve problems with clear intermediate steps and solution states, e.g., Game of 24 or 2x2 pocket cube. This model is then used to infer a graph of thought where paths leading to a solved state of the problem are converted to text and fed to an LLM as assisting knowledge to solve the initial question. The LLM is instructed to review the steps and refine them if necessary, hereby iteratively refining the solution graph by applying the first stage to replace incorrect nodes.

ResPrompt [99] is a single-prompt approach extending CoT by augmenting the few-shot examples with residual links, i.e., previously derived results. If a current reasoning step depends on previous results, these results are referenced verbatim in parentheses, effectively forming an arbitrary reasoning graph. Later stages can explicitly rely on multiple previous results as opposed to a single previous result.

C.3 Hypergraphs

Finally, we also consider a **hypergraph**, which generalizes a graph by enabling edges to connect arbitrary subsets of nodes instead of being links between just two nodes. We include hypergraphs in the taxonomy, because preliminary works already harness them for multi-modal prompting [212].

Hypergraph-of-Thought (HoT) [212] is a multi-modal reasoning paradigm modeling the thought process as a hypergraph. First, a graph-of-thoughts as in [215] is constructed. Then a textual hypergraph is constructed, sharing the same nodes. The hyperedges are then defined as node triples, e.g., "(Lionel Messi, place of birth, Rosario)". Additionally, a visual hypergraph-of-thought is constructed by performing k-means clustering on image patches, where a cluster corresponds to a hyperedge. Both hypergraphs are then encoded and combined to perform graph learning.

C.4 Analysis & Comparison of Designs

We now broadly discuss and analyze graph designs with respect to different aspects of our blueprint. A detailed analysis can be found in Appendix E.3.

C.4.1 Topology & Its Construction

Firstly, the considered schemes exhibit a blend of single- and multi-prompt aspects, allowing for a high degree of flexibility and control over the prompting process. This is evident in the diverse approaches taken by different schemes such as GoT, ControlLLM, and Cumulative Reasoning, each offering unique ways of constructing and utilizing graphs for problem-solving. Secondly, the user's control over the topology of the graph is significant in most schemes, enabling customization of the reasoning process based on specific needs, such as setting branching factors or defining the depth of the graph. Thirdly, the role of the LLM in these graph-based schemes is multifaceted, involving the generation, evaluation, and modification of nodes within

the graph, as well as determining the conclusion of the reasoning process. Lastly, there is a notable variation in the degree of user and LLM influence on the topology across different schemes, with some allowing direct user control, while others rely on predefined heuristics or the LLM's decision-making capabilities.

C.4.2 Performance

The considered works universally show improvements in effectiveness of graph-based prompting schemes over chains and trees across various tasks, suggesting a promising direction for future research and application in the field of AI and machine learning.

APPENDIX D
BENCHMARKS

In this section, we introduce benchmarks designed to test the reasoning capabilities of LLMs. These benchmarks will be instrumental in the performance analysis discussion in the following section.

D.1 Arithmetic Reasoning

Arithmetic reasoning tasks, also called math word problems, focus on narrative-based mathematical questions where the model must extract and solve numerical equations from a given text-based scenario. In the datasets, each item is presented as question-answering (QA) pairs, in which the answers are formatted as direct answers or multiple choice. Frequently used datasets include GSM8K (Grade School Math) [48], SVAMP (Simple Variations on Arithmetic Math word Problems) [151], MAWPS (MAth Word ProblemS) [113], MultiArith [161], AddSub [88], DROP (Discrete Reasoning Over the content of Paragraphs) [60] and AQUA [125].

Based on these benchmarks, datasets enhanced for more comprehensive textual understanding and more complex calculations were developed. GSM-hard [67] extends GSM8K by introducing calculations of much larger numbers. AQUA-RAT [125] requires more detailed rationales in answering than AQUA. MathQA [2] selects the QAs from AQUA and provides operation programs. MATH [86] is built on the problem sets from math competitions and includes LATEX representations. ALGEBRA [84] focuses on the algebraic domain.

Researchers take specific domains into consideration. FinQA (Financial Question Answering) [42] involves the analysis and interpretation of financial data, reports, and tables. ConvFinQA [43] presents the financial questions in a conversational manner. To test the comprehension of structured data, TabMWP [137] collects math word problems that involve tabular data.

D.2 Commonsense and Logical Reasoning

In commonsense reasoning, benchmarks test the abilities to understand text and to generate responses that align with human-like commonsense knowledge. Following that principal, group of datasets composed of multi-hop questions are constructed, including HotPotQA [211], StrategyQA [71], MuSiQue [184], Bamboogle [152], and CommaQA-E (Communicating with Agents for QA) [104]. Apart from the complexity of the questions, more QA datasets incorporating stepwise explanations for the answers are built, such as 2WikiMultiHopQA (2Wiki stands for Wikipedia and Wikidata) [87], ProofWriter [179], EntailmentBank [52]. SocialQA [172] focuses on daily social events.

Logical reasoning tasks aim to test the ability to understand contextually cause-and-effect relationships. CauseEffect [172] tasks the model with determining the causal event given descriptions of two events.

D.3 Symbolic Reasoning and Other Domains

There are several tasks involving manipulations of symbols in different domains. For Last Letter Concatenation [195], the model concatenates the last the letters of two given words into an abbreviation. Coin Clipping [195] tasks the model with determining, whether the head side of a coin faces up after a group of "flip/not flip" commands.

In the domain of spatial planning tasks, Brick World [89] asks the model to acquire a specific brick after sequentially grabbing group of bricks in 2D/3D scenarios. SCAN (Simplified version of the CommAI Navigation tasks) [115] consists of sets of compositional navigation commands paired with the corresponding action sequences. Other text-based spatial tasks are also popular, such as NLVR-based Manipulation [175] and SPARTUN [143]

Semantic parsing datasets focus on the ability of the models to generalize from knowledge about components. Examples include CFQ (Compositional Freebase Questions) [103], COGS (COmpositional Generalization Challenge based on Semantic Interpretation) [109], and Alchemy [134].

Other benchmarks are intended for special domains, such as coding (CodeNet [153]) or pedestrian traffic behavior (PIE (Pedestrian Intention Estimation) [159]) for autonomous driving.

Benchmarks that target multiple domains like MT-Bench [232] and MMLU (Massive Multitask Language Understanding) [85] measure how broad the knowledge of a model is. Sometimes just multiple areas of a single domain are targeted, for example with ScienceQA [135]. ASQA (Answer Summaries for Questions which are Ambiguous) [173] focuses on ambiquous questions in the context of long-form question answering. Several benchmarks such as ALFWorld [169] and VQA-V2 (Visual Question Answering) [76] look into visual tasks, while others target logical reasoning such as LogiQA [128] and FOLIO [82]. AutoTNLI [114] consists of counterfactual entity-based tables.

Other evaluation methods also include puzzles, like the Game of 24 (given four numbers, find the combination of those numbers and the four basic mathematical operations, so that the end result is 24), the 8-Puzzle (3x3 board with a missing piece, goal is to slide the pieces to their target location) or 2x2 pocket cube (smaller version of the Rubik's cube).

APPENDIX E

DETAILED ANALYSES

E.1 Chain

E.1.1 Performance Analysis

Chain-structured prompting methods have been applied to solve questions in diverse fields. We conduct a qualitative comparison of these methods in different domains.

E.1.1.1 **Arithmetic Reasoning**: IO prompting exhibits very low performance on mathematical tasks. Compared with IO prompting, CoT attains an accuracy of around 60% (40% higher) on GSM8K [48], 80% (10% higher) on SVAMP [151] and 90% (14% higher) on MAWPS [113]. The accuracy of the rationales significantly increases with the scale of LLMs. Zero-shot-CoT does not outperform CoT on arithmetic tasks, but obtains a 60% gain in accuracy on MultiArith [161] over IO prompting, simultaneously exceeding IO prompting on GSM8K, AQUA [69] and SVAMP tasks. Compared with CoT, PoT with few-shot examples improves on GSM8K and AQUA by more than 8% and 4% on SVAMP. For the financial datasets FinQA [42] and ConvFinQA [43], few-shot PoT reaches a roughly 20% improvement over CoT. In comparison with CoT, schemes that employ decomposition demonstrate a modest advantage on math reasoning tasks. Least-to-Most Prompting increases the accuracy on DROP [60] by 10% because of the strong decomposability of the problems in the dataset. Although Least-to-Most Prompting only shows a slight average improvement on GSM8K, it significantly enhances accuracy (by 5%) for problems within GSM8K that require more than 5 inference steps to solve. PS+ prompting yields at least 5% improvement on MultiArith, AQUA, AddSub [88], and SVAMP, but only a slight gain on GSM8K due to the high complexity and low decomposability of the problems in GSM8K. Schemes incorporating reflection nodes show limited advancement in math-related tasks, yet exhibit promising capabilities in contextual and textual comprehension. Tested on all math-related benchmarks mentioned above, PVS exhibits a 5.5% improvement on average. Moreover, PVS shows an improvement of accuracy larger than 10% on ALGEBRA [84] and GSM-hard [67] that contain algebraic calculations with large numbers, due to the integration of PoT and EoT methods. With the plug-in of the table reader tools, Chameleon increases the accuracy of answering by 8% over CoT in the math questions represented in tabular context on the TabMWP [137] dataset.

E.1.1.2 **Commonsense Reasoning**: CoT outperforms IO prompting in terms of accuracy (75.6% vs 69.4%) on the StrategyQA [71] dataset, and performs 20% better on sports understanding tasks than experienced humans. Zero-shot-CoT does not provide performance gains on commonsense reasoning benchmarks over IO prompting. The evaluation of SelfAsk focuses on multi-hop questions that require composing multiple facts to solve and improves over CoT by smaller margins on 2WikiMultiHopQA [87] and MuSiQue [184], but by 11% on Bamboogle [152] without heavy token usage. Tested on bAbI QA [97] and ProofWriter OWA [179] datasets, the SI framework enabled, on average, a 7B LLM (58.75% accuracy) to outperform a 280B LLM (44.03%) within the scope of CoT frameworks. Taking CoT as the baseline, schemes with decomposition show varying levels of performance on commonsense reasoning tasks. The problems in SCAN [115] require the conversion of single textual commands into action sequences, which suits the schemes that incorporate decomposition nodes. Here, Least-to-Most achieves 99.7% accuracy compared to just 16.2% for CoT. While PS+ shows limited improvement on StrategyQA due to the simple guidance from zero-shot prompting, Decomposed Prompting attains a 20% improvement on the CommaQA-E [104] dataset and a 30% improvement on 2WikiMultiHopQA and Musique, which stem from the designed retrieval function. With the retrieval and calculation tools, ChatCoT obtains a 20% gain on the HotPotQA [211] dataset compared with CoT.

Refinement improves incorrect thoughts, leading to higher reasoning abilities of LLMs. Reflexion's success rate on HotPotQA progressively improves with the accumulation of refinement iterations and surpasses 70% after six attempts, while CoT maintains a steady success rate of around 30%. LogiCoT also exhibits enhancement in reasoning tasks related with contextual understanding and causality inference, achieving an improvement of around 10% on the SocialQA [172] and CauseEffect [172] datasets.

E.1.1.3 **Symbolic Reasoning**: In the task of concatenating the last letters of words, CoT achieves almost 100% accuracy on cases that were part of the in-context exemplars and around 70% accuracy on out-of-domain (OOD) cases, while IO prompting shows nearly no accuracy on both cases. For coin flipping, IO prompting with a 540B LLM achieves nearly 100% on in-domain cases, but is inferior to CoT (50% vs 90%) on OOD cases. The problem-solving rate of Zero-shot-CoT is lower than CoT, but higher than IO on coin flip and last letter concatenation tasks. In the domain of spatial reasoning, CoS exhibits an increase of accuracy ranging from 2% to 40% on Brick World, NLVR-based Manipulation [175], Natural Language Navigation [78] and the spatial QA dataset SPARTUN [143], which completely exceeds the performance of CoT while using a lower number of tokens. The use of decomposition enables multi-prompt chain schemes to outperform CoT on OOD cases of the last letter concatenation task, where the length of the tested words exceeds those in the exemplars. The accuracy of the LLM gradually decreases from 94% to 74% as word length increases from 4 to 12. In contrast, CoT's accuracy drops from 70% to 32%. The accuracy of the zero-shot method PS+ is still 10% higher than few-shot CoT's. Moreover, in the more complex task of concatenating the kth letter, Decomposed Prompting shows higher accuracy (more than 96%) than CoT (around 70%) and Least-to-Most (around 12%).

E.1.1.4 **Reasoning in Special Domains**: The performance evaluation of the schemes in special domains is only selectively and qualitatively detailed here, due to the non-uniform, artificially designed metrics utilized in the assessment. Refinement schemes perform well in the code-related tasks. SELF-REFINE successfully increases readability of code on the CodeNet [153] dataset, and obtains higher speedup than the IO prompting methods on code optimization with the PIE [159] dataset. Meanwhile Reflexion achieves high accuracy in code generation on MBPP [8] and HumanEval [39].

```
User:
Question: There are a set of bricks. Brick E is on top of brick A. Now we have
    to get a specific brick. The bricks must be grabbed from top to bottom, and
    if the lower brick is to be grabbed, the upper brick must be removed first.
    How to get brick A?
Answer: Let us think step by step:

LLM:
1. To get brick A, we find E is on top of A.
2. We find E is on the top.
3. We need to remove brick E first, as it is on top of brick A.
4. Brick A is now accessible and can be grabbed.
So we get the result as E, A.
```

Listing 1: **An example implicit single-prompt chain topology, encoded with text**. It shows the CoT example of a Brick World problem from [89] with a linear sequence of four connected nodes followed by the final solution node.

E.1.2 Representations of Topology & Schedule

Listing 1 shows an example chain topology from [89].

E.2 Tree

We now broadly discuss and analyze tree designs with respect to different aspects of our blueprint.

E.2.1 Topology & Its Construction

E.2.1.1 **Multi-Prompt Schemes**: The vast majority of schemes are multi-prompt [50], [58], [91], [106], [133], [145], [148], [170], [190], [205], [213]. In all these schemes, the topology is to a certain degree dynamically constructed by the LLM (within the set boundaries of the user and the defined approach). *The role of the LLMs* can be summarised as (1) generating new child nodes, (2) evaluating given nodes and (3) deciding when we reached a final solution and reporting the results. Then, *the user to a large degree also controls the construction process*. In ToT by Long [133], the user provides a step limit as well as a checker module (rule-based or as a DNN) that decides if a reasoning step is valid or if backtracking to a previous node is necessary. ToT by Yao et al. [213] allows the user to choose the number of samples to generate at each node from which the most promising b candidates are kept for BFS, here b defines the branching factor of the tree. When using DFS, the user provides a value threshold, so when a node evaluates to a lower score than the threshold, one backtracks to the parent and continues from there. For both presented exploration schemes (BFS and DFS) the user provides an upper bound on the total number of nodes. Thought Decomposition [205] explores the tree using stochastic beam search. Hereby, the tree is constructed level by level and, in each level, k candidates are kept (the beam size) and n new nodes are generated for each candidate. The randomness in the stochastic beam search is controlled by a user-defined parameter, similarly, the user sets the temperature for generating new samples from the LLM. The total number of steps in the search, i.e., the tree depth, is capped at 16. In CoT-SC [190], the user simply provides the number of CoT samples that should be generated, resulting in a depth–1 tree. Creswell and Shanahan [50] use beam search to explore the tree, letting the user choose the number of candidate nodes to generate on each level, the beam size as well as the maximum depth of the tree. In Dynamic Least-to-Most Prompting [58], the user has no direct influence on the structure of the tree topology. The topology is entirely based on the results of the

LLMs decomposition of the input. TouT [145] has the same user parameters as ToT by Yao et al. [213], but additionally uses an uncertainty threshold for DFS to backtrack from nodes where the uncertainty gets too large. In TomT [91], the user can influence the tree topology by limiting the total number of nodes, setting a branching factor and selecting a block size k which indirectly limits the depth of the tree by generating k consecutive reasoning steps for each node instead of using one node per reasoning step as in the ordinary ToT [213]. ToC [106] limits the tree size by an upper node limit as well as setting a maximum depth. The user can choose a branching factor k which is used to dynamically select k-shot in-context examples for the generation prompt such that the appropriate number of child nodes are generated. Finally, in SoT [148] the depth–1 tree topology is only influenced by the response of the LLM (i.e., how the LLM determines the number of child nodes). The user can only indirectly influence this topology by possibly altering the node generation prompt itself.

E.2.1.2 **Single-Prompt Schemes**: Only AoT [166] exclusively uses a single-prompt tree topology. Here, in-context examples are designed such that the reasoning follows a tree structure, i.e., the examples show algorithmic reasoning steps such as trying out different solutions with DFS on the solution space, to make the LLM search for the solution in a similar tree-structured fashion. Thus, while in-context examples are provided by the user as guidance, the topology for solving the input question is constructed dynamically on-the-fly solely by the LLM.

E.2.1.3 **Beyond Single- and Multi-Prompt**: In all the above schemes, the root node holds the initial question and remaining nodes represent intermediate or final solutions. Contrarily, Tree Prompting [170] *builds its topology during an initial learning phase*, by learning the structure of a binary decision tree through training samples, with the goal of text classification. During inference, the tree structure is fixed and independent of the specific input. The user can indirectly influence the tree topology by manually providing the prompt-candidates used for learning the binary decision tree at the training stage.

E.2.2 Performance

We perform a qualitative performance evaluation of different tree schemes. We consider what problems or datasets the approach is evaluated on, how the quality of answers scales with respect to the number of prompts (or tokens), and what the trade-offs in quality are with respect to different tree topologies or reasoning schedules.

E.2.2.1 **Considered Problems & Datasets**: ToT by Long [133], ToT by Yao et al. [213], AoT [166], and TouT [145] evaluate their performance on problems that can be simply decomposed in a tree-structured way, such as 5x5 Sudoku puzzles in ToT by Long, and Game of 24 or 5x5 Crossword puzzle for the other three approaches. Thought Decomposition [205] and CoT-SC [190] perform their evaluation on arithmetic reasoning, e.g, GSM8K [48], symbolic reasoning, e.g., object counting or last letter concatenation, as well as on commonsense reasoning tasks such as CommonsenseQA [180] and StrategyQA [71]. The scheme of Creswell and Shanahan [50] assumes access to an existing context to reason on and evaluate their approach on

ProofWriter [179] and EntailmentBankQA [52]. Dynamic Least-to-Most Prompting [58] is evaluated on semantic parsing datasets (CFQ [103] and COGS [109]), TomT [91] create a new synthetic visual question answering dataset for evaluation by combining two existing datasets, ToC [106] is evaluated on a long-form QA dataset (ASQA [173]), Tree Prompting [170] uses 13 text classification datasets and SoT [148] is evaluated on diverse questions from [46], [207].

E.2.2.2 **Accuracy vs. #prompts, topology variant, & reasoning schedule**: Long [133] does not discuss performance evaluation and the scheme is executed until a solution is found or a maximum of 100 prompts are issued. Yao et al. [213] evaluate their approach on Game of 24 using BFS on a tree of depth 3 with a branch factor of 5 and scoring each node 3 times, on Creative Writing using a tree of depth 2, branch factor of 5 and selecting the best node in each level by scoring 5 times, and finally on solving 5×5 Crosswords with a ToT of depth 5 to 10 using DFS, a branch factor of 5 and at most 100 DFS steps. Thought Decomposition [205] performs beam search on a ToT with branching factor 16 and beam size 5, essentially generating $5 \cdot 16 = 80$ samples at each tree level, while the depth is limited to 16. The authors find out that increasing the branching factor leads to more diversity and thus improves performance when using majority voting on the last level. This increase comes at the cost of using more tokens. In CoT-SC [190], the authors evaluate the number of reasoning chains to sample, i.e., using a tree of depth one with a branching factor of 1 up to 40 while reporting the average over 10 runs. Sampling more outputs improves accuracy, but also increases computational cost; most gains are realized after sampling 5 to 10 CoT-paths in practice. The approach of Creswell an Shanahan [50] assumes access to a context, i.e., a set of statements, to reason on. The individual LMs (selection, inference, two different halter modules, and correctness prediction of a step) need to be fine-tuned separately in advance, which makes this approach very specific and incurs a setup cost. Dynamic Least-to-Most Prompting [58] comes with no explicit discussion about the number of tokens used, but the number of prompts varies with the decomposition of the specific input phrase and so does the number of in-context exemplars. The authors claim a speedup of over a factor 2 compared to CoT-SC prompting as only a single result is generated and no majority voting is necessary as in CoT-SC. AoT [166] uses a single prompt for the Game of 24 (reporting 9% out of token errors) and two prompts for the 5x5 Crossword puzzle. The authors claim that ToT uses many more prompts, i.e., over 100 rsp. 200, for the two problem setups. Hereby, AoT is reported to perform better than ToT for the Game of 24. The authors show that the impact of more exploration steps inside the examples leads to longer prompt generations, i.e., slower inference for the same number of games. TouT [145] sample 20 LLM responses per ToT node to compute local variances and claim improved performance to ToT for both Game of 24 and 5x5 Crossword puzzles and further increasing or decreasing the number of samples leads to degrading performance. In TomT [91], the DFS is limited to 30 steps with their ToT-One-Stop and the reference ToT implementation using a branching factor of 3. The authors report improved accuracy for ToT-One-Stop compared to ToT while using approximately half of the number of steps.

ToC [106] claims comparable performance to Cot-SC with using less than 20 LLM calls. According to the published results, Tree Prompting [170] enables smaller models to outperform larger models and the accuracy can be improved at the cost of more LLM calls; the number of LLM calls for the experiments is limited to 40. Finally, SoT [148] reports end-to-end latency speedups across different task categories and LLMs. The authors compare the two step decoding process of SoT, i.e., creating short bullet point style answers which are expanded in the second step, against "normal decoding", where a continuous answer is generated in one go. There are a varying number of bullet points generated with the average being 9 bullet points. The authors assess the answer quality with the help of two LLM judges (FastChat and LLMZoo) using different metrics. SoT reports a high token overhead of roughly 60x to 90x, which can be optimized down to roughly 30x.

E.3 Graph

We now broadly discuss and analyze graph designs with respect to different aspects of our blueprint.

E.3.1 Topology & Its Construction

Most graph schemes combine to a certain degree single- and multi-prompt aspects. GoT by Besta et al. [10] requires a problem specific graph of operations for solving a problem in a multi-prompting fashion. One has complete freedom over the individual prompts to the LLM and can incorporate arbitrary single-prompting schemes. GoT by Lei et al. [119] dynamically constructs the graph, at the time of this writing no code or prompt templates have been published, so it is not clear to what extend single-prompting is used. GoT by Yao et al. [215] uses a NLP approach to construct a graph of thought and uses an LM as part of the AI-pipeline to learn how to use these graphs of thought for predicting an answer. ControlLLM [132] is similar to GoT [10] in the sense that all nodes in the *Tool Graph* correspond to nodes in the GoT with access to tools. Cumulative Reasoning [224] is a conceptual extension of ToT with access to all previously derived results, i.e., nodes, resulting in a DAG. XoT [57] uses a problem-specific pre-trained RL-model to create a graph of thought and uses the LLM to review and summarize the results.

In graph schemes, *the user has significant control over the topology and its creation*. In BSM [162], the topology is a double tree with one intermediate level. The user can set either the number of branches, or its upper bound by specifying the number of sub-problems to generate inside a single prompt. In Thought Propagation [218], the user specifies the branching factor in a single prompt to generate analogous problems, and can predefine the number of levels of the double tree to grow the topology. The user in Socratic Questioning [154] defines the branching factor of the topology by specifying its value, lower limit, or upper limit within a single prompt. Furthermore, the user can limit the growth of the topology by defining the maximum tree depth. In Graph of Thoughts (GoT) [10], the user defines the complete topology of reasoning across multiple prompts for a given use case, including, but not limited to, the branching factor, the maximum depth of tree, and

the number of child nodes to keep. The topology in Graph of Thought (GoT) [119] is influenced by a user-defined depth limit. In Cumulative Reasoning [224], the user can influence the total number of nodes as the proposer (LLM) iteratively generates a new node until a user-specified limit is reached. In ControlLLM [132], the user does not define the number of nodes during the task decomposition stage for solving a given task. However, a user may have implicit influence on the number of nodes via prompting, e.g., by adding the phrase "Parse out as few tasks as possible". The user in Everything of Thoughts (XoT) [57] can indirectly influence the topology by defining the number of times the LLM is prompted to review and refine the solution graph. In ResPrompt [99], the user simply provides in-context examples which directly dictate the basis of the topology. Differing from previous schemes, users in two-stage frameworks such as Graph-of-Thought (GoT) [215] and Hypergraph-of-Thought (HoT) [212] do not influence the topology as the (hyper)graph-of-thought is constructed by predefined heuristics.

Similar to prompting with trees, *the LLM may also influence the topology of prompting with graphs* within user-defined boundaries. The role of the LLM can be summarized as (1) generating new child nodes, (2) evaluating given nodes, (3) deleting or backtracking from a new node (e.g., based on evaluation results), and (4) determining when the reasoning process should be concluded.

E.3.2 Performance

E.3.2.1 **Considered Problems & Datasets**: BSM [162] evaluates its performance on answering questions from multiple areas by using MT-Bench [232]. TP [218] performs shortest-path reasoning, creative writing and LLM-Agent Planning on ALFWorld [169]. Socratic Questioning [154] evaluates their method on both, text-only datasets (e.g. MMLU [85] and MATH [86]), logical reasoning questions (LogiQA [128]), as well as multi-modal visual QA datasets such as VQA-V2 [76] and two others. GoT by Besta et al. [10] performs integer sorting with duplicates, set intersections, keyword counting and document merging. GoT by Lei et al. [119] measures its performance on the Game of 24, solving higher-degree polynomial equations as well as deriving formulas for recursive sequences. GoT by Yao et al. [215] evaluates on the arithmetic reasoning benchmark GSM8K [48] and on ScienceQA [135], which contains science questions with images. ControlLLM [132] provides its own benchmark containing tasks for image editing and perception, visual QA and the usage of over 20 different common tools. Cumulative Reasoning [224] evaluates its approach on FOLIO [82], AutoTNLI [114], Game of 24, as well as mathematical reasoning on the MATH [86] dataset. XoT [57] evaluates on Game of 24, 8-Puzzle and on a 2x2 pocket cube. ResPrompt [99] uses multiple math reasoning benchmarks for evaluation, i.e., GSM8K [48], AQUA-RAT [125], MathQA [2] and SVAMP [151], as well as the sequential reasoning benchmark SCONE-Alchemy [134] and commonsense reasoning questions from StrategyQA [71]. HoT [212] evaluates their approach on ScienceQA [135].

In summary, there is no clear set of benchmarks that is used consistently with all approaches, but most include math reasoning problems or some common reasoning tasks such as Game of 24.

E.3.2.2 **Accuracy vs. #prompts, topology variant, & reasoning schedule**: BSM [162] uses a branching factor of 5 for its experiments and outperforms zero-shot prompting and CoT-SC (sampling 5 times). BSM makes LLaMA-2-70B-chat competitive with zero-shot GPT-4 for the turn-1 questions. BSM has similar computational requirements as CoT-SC, but yields higher scores. Additionally, BSM benefits smaller models (LLaMA-2-7B), where CoT-SC is ineffective. The benefits of increasing the branching factor seems to be saturated at 4, but depends on the specific problem. TP [218] for shortest-path reasoning makes significant performance gains over all the baselines (IO, CoT, ToT) for all evaluated models, i.e., PaLM-2, GPT-3.5 and GPT-4. The authors report that there is only a marginal difference between 1-shot and 5-shot prompting without further exploration. The performance can directly be influenced by the number of layers used for TP. Hereby, 1-layer TP has competitive performance to ToT with a similar amount of tokens used. Socratic Questioning [154] is compared against IO, CoT, CoT-SC and ToT prompting. The authors report 3-4% absolute gain over other methods. The reported ToT accuracy is relatively low compared to the other baselines. GoT by Besta et al. [10] significantly improves upon IO, CoT, and ToT on all four tasks using GPT-3.5. The accuracy enhancement of GoT over the best-performing baseline (ToT) is achieved with lower or comparable number of tokens. The performance advantages of GoT over the baselines increase with the problem size.