Recursive Decomposition with Dependencies for Generic Divide-and-Conquer Reasoning

Aalto University

Sergio Hernández-Gutiérrez Minttu Alakuijala Alexander V. Nikitin Pekka Marttinen

Overview

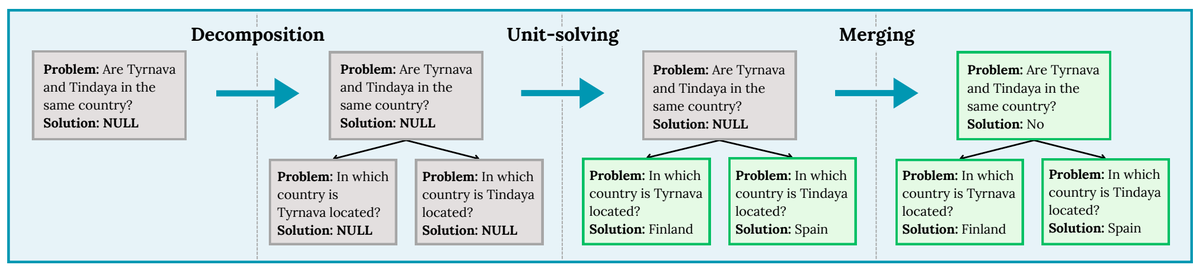
We introduce **Recursive Decomposition with Dependencies (RDD)**, a scalable divide-and-conquer method for solving reasoning problems with Large Language Models (LLMs)

- consists of the following steps: **decompose** → **schedule** → **solve recursively** → **merge**
- requires **less supervision** than prior decomposition approaches
- directly applicable to a new problem class **without task-specific guidance**
- supports sub-task dependencies, ordered execution of sub-tasks and **error recovery** from mistakes made in previous steps
- **outperforms** other methods in a compute-matched setting **as task complexity increases**, while also being more **computationally efficient**

Recursive Decomposition with Dependencies (RDD)

We will refer to the initial problem provided by the user as the *root problem* $x_0 \in T^l$, where T is a set of tokens, and l is the length of the prompt. Our method consists of three steps:

1. **Decomposition:** The root problem is recursively decomposed into sub-problems by prompting the LLM with the decomposition meta-task. The model generates either a list of sub-problems or the response "*This is a unit problem.*"
2. **Unit-solving:** Unit cases can be solved with either direct input-output prompting, any other reasoning method, or an external tool.
3. **Merging:** For each set of already-solved sub-problems, we prompt the LLM to merge their solutions to solve their parent problem; we perform this process until we reach the root problem, at which point we obtain the final solution to x_0 via the last merging step. The model is also encouraged to fix any erroneous sub-solutions at merging time.



Modeling sub-problem dependencies: We ask the model to assign a unique ID to each sub-problem. We also encourage the model to cross-reference solutions of other sub-problems by their IDs. The specified dependencies correspond to a directed acyclic graph (DAG).

Scheduler: The scheduler defines the execution order of the decomposition, unit-solving, and merging steps (with ScheduleBFS). For a parallelized implementation, the scheduler synchronizes the execution of dependent sub-problems.

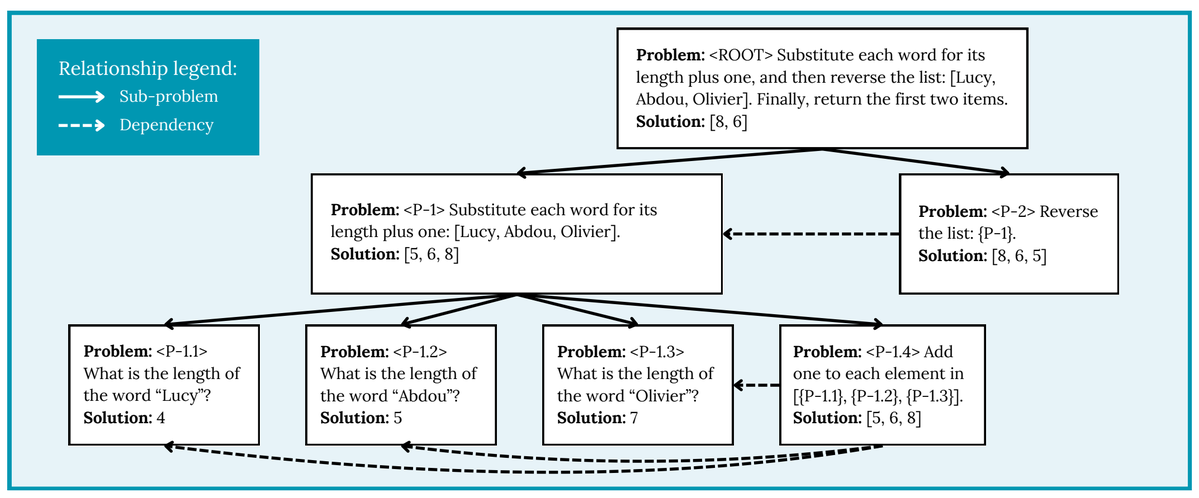
Information flow: When prompting for a decomposition, only the current problem description is provided to the model, along with a description and a set of demonstrations of the meta-task (e.g., decomposition or merging). We do not include the history of ancestor problem descriptions, which increases in size with the depth of the recursion process. In the merging step, the decomposition of the current level and its sub-solutions are provided.

Algorithm 1 ScheduleBFS

Input: problem

```
unsolved ← empty queue
// If problem depends on other sub-tasks, solve them first
for dependency ∈ problem.dependencies do
    ScheduleBFS(dependency)
end for
// Decompose the input problem
sub-problems ← Decompose(problem)
for sub-problem ∈ sub-problems do
    Add sub-problem to unsolved
end for
// Decompose each sub-problem
while unsolved is not empty do
    next-problem ← unsolved.front
    for dependency ∈ next-problem.dependencies do
        ScheduleBFS(dependency)
    end for
    sub-problems ← Decompose(next-problem)
    for sub-problem ∈ sub-problems do
        Add sub-problem to unsolved
    end for
end while
// Apply unit-solving and merging with depth-first-search
return ScheduleDFS(problem, [ ])
```

Recursive decomposition of a problem with dependencies



Related works

Decomposition as a tree: Tree of Thoughts [1] builds a tree; however, it represents a sampling process, not a recursive decomposition. Divide-and-Conquer prompting [2] applies a tree-like recursive decomposition strategy, without supporting sub-problem dependencies.

Decomposition as a graph: Although DecomP [3] also implicitly models the solving process as a DAG via tool usage, it performs steps in sequence in a chain-like fashion (preventing parallelization), and its structure needs to be demonstrated by the user for every problem class, and it does not model dependencies between sub-problems. Graph of Thoughts [4] explicitly uses a DAG, but both its structure and the meaning of the nodes must be provided by the user for every problem instance.

Generic applicability: Reasoning methods resulting in complex computational graphs [1, 3, 4] often require extensive user-generated input to model the reasoning process. In contrast, RDD can model complex reasoning structures without unrealistic data requirements at runtime.

Parallelization capabilities: Similar to Skeleton-of-Thought (SoT; [5]), we enable efficient decoding by identifying independent reasoning steps that can be computed in parallel. However, SoT does not decompose reasoning chains. Sequential decomposition methods [3] do not support parallelization.

Experiments

Setup: We compare against Chain-of-Thought (CoT; [6]) and Least-to-Most prompting (LtM; [7]). We use binary search self-consistency (SC; [8]) to align the amount of computation available to each method. We consider benchmark tasks of increasing difficulty. The tasks are:

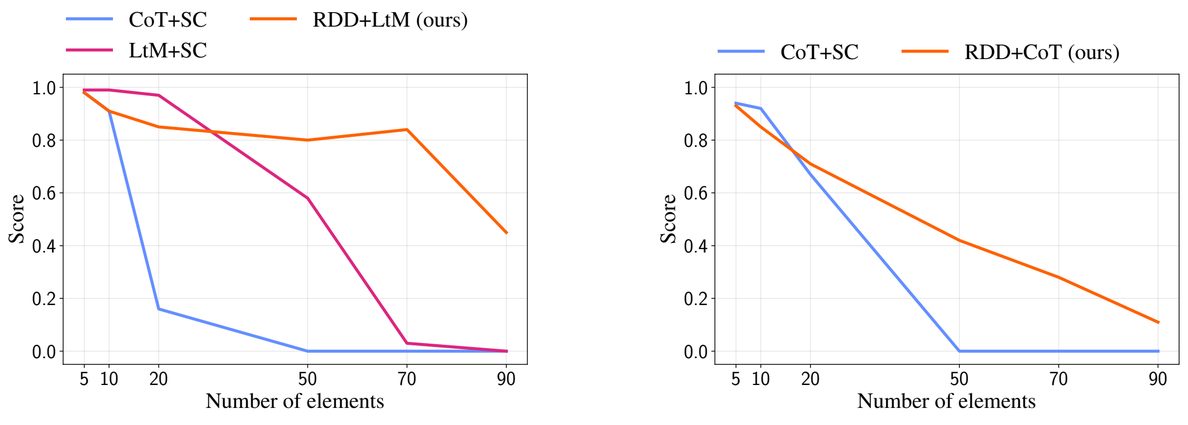
- **Letter concatenation:** concatenate the character at position *i* in every string of the input array.
- **Length reversal:** replace each string in the input array for its length, and reverse the resulting array.

Transition points: We define the functions predicting the accuracy of the decomposition, unit-solving, and merging steps as $\phi_{\text{d}}(X_0), \phi_{\text{u}}(X_0), \phi_{\text{m}}(X_0) \in [0,1]$, respectively, where X_0 is a random variable from the domain $\mathcal{P}_{c_0,n_0}$, the set of root problem instances x_0 belonging to problem class c_0 and with difficulty of n_0. We then define $\phi_{\text{RDD}}(X_0)$ to be the overall accuracy of RDD.

We hypothesize the existence of a performance transition point at within-class difficulty n^*, after which $\phi_{\text{RDD}}(c_0, n_0) \geq \phi_{\text{u}}(c_0, n_0)$ will hold $\forall n_0 \geq n^*$. We empirically observe such transition points.

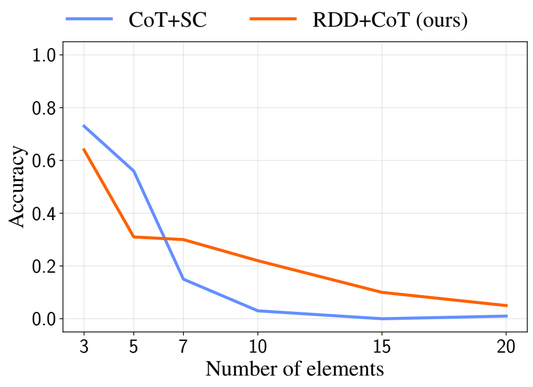
Comparison between RDD, CoT+SC and LtM+SC

RDD uses CoT or LtM at the unit case. All reported numbers are averages over 100 input problems.



(a) Letter concatenation (**task-specific examples**)

(b) Letter concatenation (**generic examples**)



(c) Length reversal (**generic examples**)

Conclusions

Through our empirical evaluation, we have demonstrated the following advantages:

- Our approach, RDD, **increases accuracy in complex reasoning problems** over state-of-the-art methods in a compute-matched setting.
- The recursive decomposition technique augments the model's reasoning abilities even **without any task-specific data**.
- Our method reduces the time to reach a solution by generating fewer output tokens; parallelization enables a further speedup.
- RDD reduces the average number of tokens per call, lessening strain on the context window.

References

[1] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R. Narasimhan. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *NeurIPS*, 2023.

[2] Yizhou Zhang, Lun Du, Defu Cao, Qiang Fu, and Yan Liu. An examination on the effectiveness of divide-and-conquer prompting in large language models. *ArXiv*, 2024.

[3] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed Prompting: A Modular Approach for Solving Complex Tasks. In *ICLR*, 2022.

[4] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of Thoughts: Solving Elaborate Problems with Large Language Models. In *AAAI*, 2024.

[5] Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. Skeleton-of-Thought: Prompting LLMs for Efficient Parallel Generation. In *ICLR*, 2023.

[6] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *NeurIPS*, 2022.

[7] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V. Le, and Ed H. Chi. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. In *ICLR*, 2022.

[8] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *ICLR*, 2022.

NEURAL INFORMATION PROCESSING SYSTEMS