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

Anonymous Author(s)
Affiliation
Address
email

Abstract

Reasoning tasks are crucial in many domains, especially in science and engineering. Although large language models (LLMs) have made progress in reasoning tasks using techniques such as chain-of-thought and least-to-most prompting, these approaches still do not effectively scale to complex problems in either their performance or execution time. Moreover, they often require additional supervision for each new task, such as in-context examples. In this work, we introduce Recursive Decomposition with Dependencies (RDD), a scalable divide-and-conquer method for solving reasoning problems that requires less supervision than prior approaches. Our method can be directly applied to a new problem class even in the absence of any task-specific guidance. Furthermore, RDD supports sub-task dependencies, allowing for ordered execution of sub-tasks, as well as an error recovery mechanism that can correct mistakes made in previous steps. We evaluate our approach on two benchmarks with six difficulty levels each and in two in-context settings: one with task-specific examples and one without. Our results demonstrate that RDD outperforms other methods in a compute-matched setting as task complexity increases, while also being more computationally efficient.

1 Introduction

Large language models (LLMs) have been proven successful as the backbone of generic intelligent systems (OpenAI, 2022, 2024; Anil et al., 2023; Anthropic, 2024). These models possess strong conversational skills (Radford et al., 2019; Brown et al., 2020; Chowdhery et al., 2023; Touvron et al., 2023), making them an effective tool to interact with users in a wide range of settings. However, the autoregressive architecture of transformer-based language models limits the complexity of problems that can be solved. As a result, language models struggle with reasoning tasks (Nogueira et al., 2021; Deletang et al., 2022; Dziri et al., 2023; Chen et al., 2023), from multi-hop question-answering and symbolic manipulation to arithmetic and logical inference. Recent methods have been proposed to increase the performance of LLMs on reasoning problems (Wei et al., 2022; Wang et al., 2022b; Zhou et al., 2022; Yao et al., 2023; Besta et al., 2024; Khot et al., 2022). In particular, many of these techniques focus on eliciting step-by-step solving processes or decomposition strategies.

Nonetheless, we identify three issues affecting the currently available approaches. First, previous methods typically build a single reasoning chain (Wei et al., 2022; Wang et al., 2022b; Zhou et al., 2022; Khot et al., 2022; Yao et al., 2023), without supporting independent, parallelizable sub-tasks, and allow for limited or no communication between alternative chains. When decomposition is supported, prior work has been limited to fully separable decompositions, without supporting dependencies between sub-tasks (Zhang et al., 2024). Second, existing methods often require the user to provide task-specific examples (Khot et al., 2022) or a pre-defined decomposition strategy (Zhou et al., 2022; Zhang et al., 2024; Besta et al., 2024), making them difficult to incorporate into generic intelligent systems. Third, the number of tokens required to express their reasoning chain frequently scales quadratically with respect to the complexity of the task at hand (Zhou et al., 2022), which becomes an even more pressing issue when considering the limited context window and high computational cost of LLMs. Additionally, as we empirically show, their downstream performance decays rapidly with increasing task complexity. Our work addresses these issues, empowering LLMs to solve more complex reasoning problems with decomposition strategies that are readily applicable to real-world generic intelligent systems.

We propose Recursive Decomposition with Dependencies (RDD), a flexible and task-agnostic framework for task decomposition with desirable scaling properties and high potential for parallelization. Specifically, we use in-context learning to decompose reasoning problems into sub-problems, solve these individually, and then merge their solutions to solve the original problem. The model can optionally model dependencies between the sub-tasks proposed during the decomposition step.These steps are applied recursively: sub-tasks are repeatedly broken down until either a base case is reached or specific stopping criteria are met. By incorporating relevant in-context examples, sub-task indices, and a scheduler, our method can automatically generate new sub-tasks. It does this by using the results of completed sub-tasks as input for others, allowing the decomposition structure to extend from a simple tree to a more complex directed acyclic graph (DAG). These generic operations (`split`, `solve`, and `merge`) are applicable across tasks and do not require user intervention. Our decomposition strategy reduces the strain on the context window of the model and shortens the runtime per input problem.

Our contributions are:

1. We introduce a novel method, Recursive Decomposition with Dependencies (RDD), for solving reasoning problems (Sec. 2) via decomposition into smaller subtasks with dependencies,
2. We empirically demonstrate the effectiveness of our approach, both with and without task-specific demonstrations (Sec. 3),
3. We evaluate our method on one task decomposable into independent sub-problems and another requiring dependency modeling (Sec. 3).

2 Recursive Decomposition with Dependencies

We assume access to a pretrained LLM and leverage in-context learning and prompting strategies to implement RDD. Our method consists of three steps: *decomposing*, *unit-solving*, and *merging*. 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. **Decomposition:** The root problem is initially 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*." If the problem is identified to be a unit case, RDD prompts the model to solve it directly. Otherwise, the decomposition meta-task is repeated with each of the sub-problems recursively. **Unit-solving:** Unit cases can be solved with either direct input-output prompting, any other existing reasoning method, or by employing an external tool. **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. A visual representation of this procedure is provided in Fig. 1 for a non-recursive case of depth one.

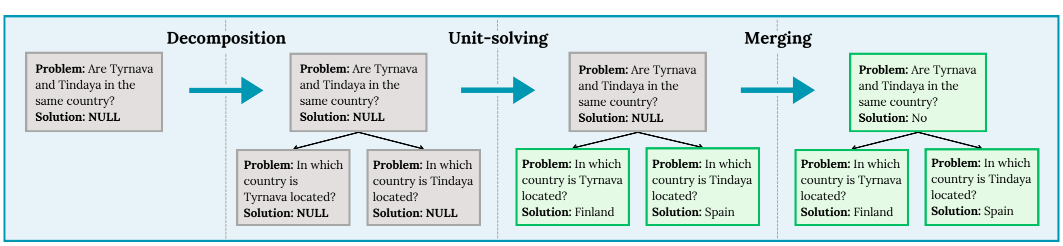


Figure 1: The decomposition methodology pipeline: decomposing, unit-solving, and merging. Nodes in gray represent unsolved problems, while nodes in green represent solved problems.

2.1 Notation and definitions

We estimate the conditions under which applying RDD is beneficial compared to a direct solution attempt by estimating the success rates of the decomposition, unit solving, and merging steps. We can define the function predicting the accuracy of the decomposition step as $\phi_{\mathrm{d},\mathcal{M}_\theta,\mathcal{C}}(c,n) \in [0,1]$, and similar functions for the unit-solving and merging steps as $\phi_{\mathrm{u},\mathcal{M}_\theta,\mathcal{C}}(c,n) \in [0,1]$ and $\phi_{\mathrm{m},\mathcal{M}_\theta,\mathcal{C}}(c,n) \in [0,1]$, respectively, where c is the input problem class (e.g., multiplying two numbers), n is the within-class difficulty of the input problem (a problem-specific metric; typically, the size of the input data), $\mathcal{M}_\theta$ is a language model which executes the decomposition, unit-solving and merging steps, and $\mathcal{C}$ is a classifier (in our method, implemented by $\mathcal{M}_\theta$) which returns `true` if a (c,n) pair constitutes a unit problem and `false` otherwise. We may refer to the within-class difficulty also as just *difficulty*. Other variables, such as the maximal branching factor or *width* w of each decomposition step, may influence the expected accuracy of RDD, but are treated as fixed constants in this notation. We may also omit $\mathcal{M}_\theta$ and $\mathcal{C}$ for conciseness, assuming them to be constant. We define ϕ_{RDD} to be the overall accuracy of RRD applied with a model $\mathcal{M}_\theta$ and classifier $\mathcal{C}$ on a problem of class c with difficulty n. We can also relax our existing notation for all aforementioned functions to only depend on a given random variable X_0 from the domain $\mathcal{P}_{c_0,n_0}$, the set of root problem instances x_0 belonging to class c_0 and with difficulty of n_0. We can approximate the individual and overall expected accuracies empirically by measuring their success rates for a large enough set of problem instances, which we explore in Sec. 3.4 within the scope of our evaluation setting.

Transition points Let c_0 be a fixed problem class solvable by $\mathcal{M}_\theta$. We expect that, as we lower the within-class difficulty n_0 of the problem instances X_0 of class c_0, we get $\phi_{\mathrm{u}}(X_0) \lessapprox 1$. In such cases, we expect $\phi_{\mathrm{RDD}}(X_0) \leq \phi_{\mathrm{u}}(X_0)$, since decomposing X_0 will likely not improve the accuracy of the unit cases sufficiently to compensate for the additional decomposition and merging steps. Thus, we hypothesize the existence of a performance transition point at within-class difficulty n^*, after which $\phi_{\mathrm{RDD}}(c_0,n_0) \geq \phi_{\mathrm{u}}(c_0,n_0)$ will hold $\forall n_0 \geq n^*$. We empirically observe such transition points.

2.2 Methodology

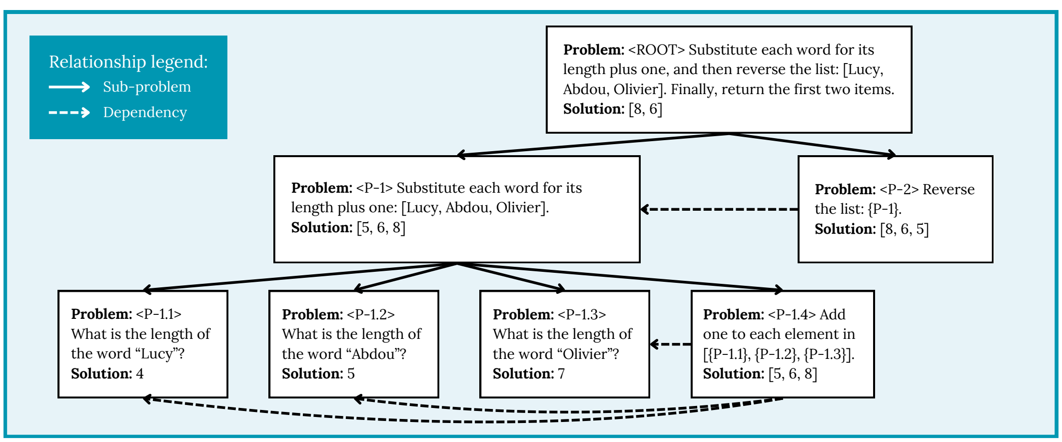


Figure 2: An example of the decomposition graph generated by the RDD method.

RDD enables the language model $\mathcal{M}_\theta$ to suggest dependencies between sub-problems in the decomposition step. We request the model to assign a unique identifier (e.g., "*P-1*" and "*P-2*") to each proposed sub-problem. We also encourage the model to cross-reference solutions from other sub-problems via their identifiers (e.g., "*Reverse the following list: {P-1}*"). This construction implies that dependent problems cannot be decomposed nor solved without first solving their dependencies. Using the dependencies specification generated by the model, we can now define edges between sub-problems with a common parent in the decomposition tree, resulting in a directed acyclic graph (DAG). Fig. 2 shows an example decomposition graph produced by RDD.

Information flow When solving sub-problems, we aim to minimize the amount of information about ancestor tasks included in the context to isolate the relevant information and achieve better scaling properties with respect to the difficulty of the root problem. 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; this feature requires a stronger language model to always provide all needed data and instructions in the description of each sub-problem. In the merging step, the decomposition of the current level and its sub-solutions are provided.

Maximizing generic applicability We consider a fixed set of generic meta-task demonstrations included in the decomposition, merging, and unit-case prompts. This set exhibits a diverse range of tasks. We show experimentally that this same set of generic examples is effective for guiding decomposition for new, unseen tasks. The examples we use in our evaluation are available in App. D. Moreover, the meta-tasks RDD performs are fixed, regardless of the input problem, and thus also task-invariant. The generality of the demonstrations is a spectrum and thus presents a trade-off between the degree of applicability of the methodology and the degree of assistance it provides to $\mathcal{M}_\theta$, the latter variable being correlated with performance. To increase performance in a given domain (e.g., programming assistance) at the cost of generality, the generic demonstrations can be selected from the same domain (e.g., coding problems).

Scheduler The scheduler defines the execution order of the decomposition, unit-solving, and merging steps. The order of decomposition defines the structure of the resulting graph. The root problem is expanded via breadth-first search (BFS) traversal until an unsolved dependency is found, in which case the current traversal process halts and executes the BFS routine with the dependency as the root problem. A depth-first search (DFS) traversal schedules the unit-solving and merging steps. The complete procedure we employ is explicitly reflected in Algorithm 1. The SCHEDULEDFS procedure is provided in App. B; the DECOMPOSE routine corresponds to the decomposition step. An example execution of this algorithm shown in Fig. 2 is provided in App. I. For a parallelized implementation, the scheduler synchronizes the execution of sub-problems with inter-dependencies.

Algorithm 1 SCHEDULEBFS

```
Input: problem
unsolved ← empty queue
for dependency ∈ problem.dependencies do
    SCHEDULEBFS(dependency)
sub-problems ← DECOMPOSE(problem)
for sub-problem ∈ sub-problems do
    Add sub-problem to unsolved
while unsolved is not empty do
    next-problem ← unsolved.front
    for dependency ∈ next-problem.dependencies do
        SCHEDULEBFS(dependency)
    sub-problems ← DECOMPOSE(next-problem)
    for sub-problem ∈ sub-problems do
        Add sub-problem to unsolved
return SCHEDULEDFS(problem, [ ])
```

3 Empirical Evaluation

The hypotheses we aim to validate through our empirical study are the following:

- **Hypothesis 1**: RDD increases accuracy in complex reasoning problems over state-of-the-art methods in a compute-matched setting.
- **Hypothesis 2**: The recursive decomposition technique augments the model's reasoning abilities even in the absence of task-specific data.
- **Hypothesis 3**: Solving reasoning problems via RDD reduces the time taken to reach a solution compared to solving the entire problem via step-by-step prompting strategies.
- **Hypothesis 4**: RDD reduces the average amount of tokens per generation process, thus lessening the strain on the context window.

As baselines, we consider Chain-of-Thought (CoT; Wei et al. (2022)) and Least-to-Most prompting (LtM; Zhou et al. (2022)). We use self-consistency (SC; (Wang et al., 2022b)) to align the amount of computation between our method and the baselines. In our implementation of SC, we employ the LLM itself to decide the most consistent answer given the set of sampled solutions. The first solution candidate is sampled greedily, while the rest are sampled with a temperature of 0.7 to produce a variety of reasoning chains. Given that each SC sample will produce a large number of generated tokens and including them all in a single context window can be challenging, we propose to use binary search to find the most consistent answer. RDD employs a single CoT or LtM chain at the unit-solving prompt but does not use self-consistency to aggregate multiple answers. We evaluate all methods on benchmark tasks of increasing difficulty. For each difficulty, scores are averaged across the same 100 randomly sampled problem instances for all methods. We employed the instruction-tuned Llama 3 70B (Meta, 2024) as the underlying model. This model was run on NVIDIA A100 and H100 GPUs. App. E provides resource usage statistics for all experiments in this section. It is additionally possible to parallelize the solving of independent sub-problems for a speedup.

3.1 Task-specific Experiments

We first experiment with task-specific examples to validate our approach. Each call to the model (i.e., all baseline calls, as well as the decomposition, unit-solving, and merging steps) operates in a 5-shot in-context setting. These examples are in-distribution with respect to the problem class c_0, but out-of-distribution with respect to the within-class difficulty n_0. For this experiment, we evaluate on the letter concatenation problem, which asks the LLM to concatenate the i'th character of each word in a list. This task can be recursively decomposed into independent sub-problems, thus not requiring dependency modeling. The difficulty n_0 is the number of words in the list.

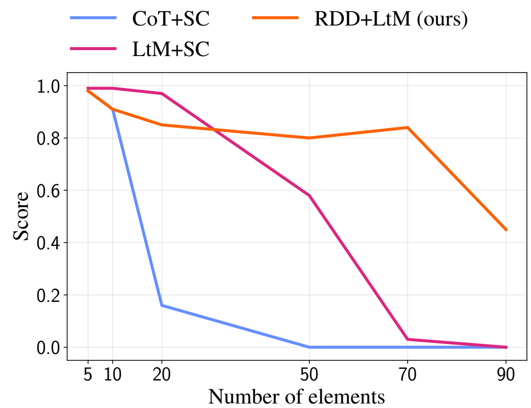
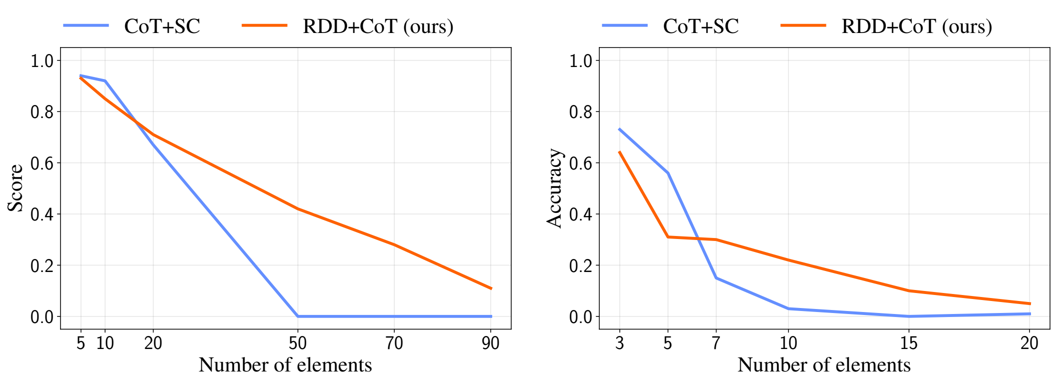


Figure 3: An evaluation of RDD against CoT (Wei et al., 2022) and LtM (Zhou et al., 2022) with self-consistency (SC; Wang et al. (2022b)) on the letter concatenation benchmark in the task-specific few-shot setting. Our system uses LtM at the unit-solving step; we refer to it as RDD+LtM.

Fig. 3 demonstrates the results for six input sizes. The score is computed as the average of an exact match metric. We find RDD to outperform the baselines as the task complexity increases, confirming Hypothesis 1. For $n_0 < 20$, it does not seem beneficial to recursively decompose the problem. We observe a transition point $20 < n_0^* < 50$ with respect to LtM+SC. In Table 1 (App. E), we show that RDD also reduces execution time with respect to the baselines.

3.2 Generic Experiments



(a) Letter Concatenation

(b) Length Reversal

Figure 4: An evaluation of RDD against CoT with self-consistency (SC) the generic few-shot setting. Our system uses CoT at the unit-solving step; we refer to it as RDD+CoT.

To evaluate Hypothesis 2, we verify whether the advantage of RDD is maintained when task-specific in-context examples are replaced with generic examples. These generic examples depict a wide range of tasks: arithmetic, coding, symbolic manipulation, multi-step logic reasoning, and multi-hop QA, but exclude the tested problem class c_0. In this experiment, we operate in a 5-shot setting for the unit-solving and merging steps, and a 7-shot setting for the decomposition step. We provide the included examples in App. D. We also improved the description of the problem with respect to the one used for the experiments described in Sec. 3.1: the model is tasked with concatenating each character using a space as a delimiter. If the characters are concatenated without a special separator, their tokenization may change when encoding them for subsequent steps in the autoregressive generation process. This behavior has been also described by Mirchandani et al. (2023) for tasks represented in grids of numbers. The rest of the setup is the same as previously described.

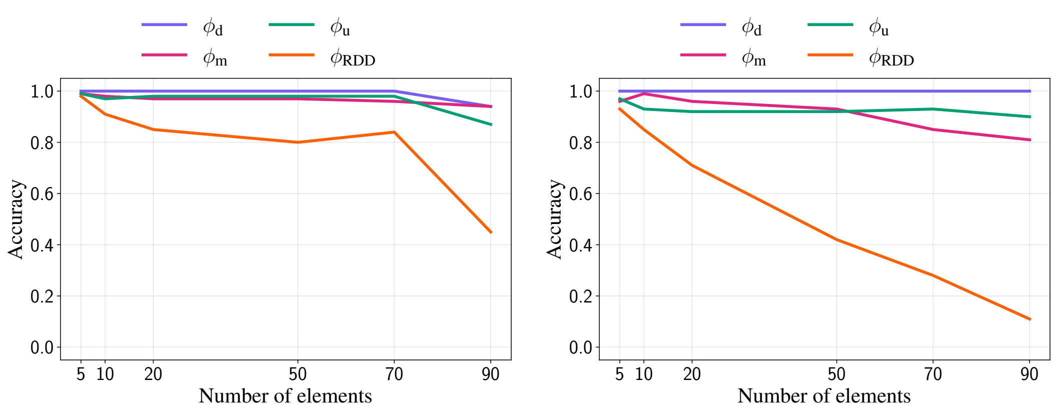
For this set of experiments, we compare RDD with CoT as the unit-solving method (RDD+CoT) against CoT with self-consistency (CoT+SC). In Fig. 4a, we can observe that RDD again outperforms this baseline as the difficulty of the task increases. We see a performance transition point $10 < n_0^* < 20$ with respect to CoT+SC. We again observe considerable time savings; a complete account of resource usage can be found in Table 2 (App. E).

3.3 Sub-tasks with Dependencies

This section shows the results of our evaluation with RDD on the task of length reversal. To solve this task, the model must substitute each word in a list with its length (number of characters), and then reverse the order of the items in the list. This task benefits from dependency modeling in the decomposition step. We compare our method against CoT+SC with generic in-context examples. We use five examples for the unit-solving and merging steps and eight for the decomposition step. The examples showcase several different decomposition and merging patterns and also cover a wide range of problem classes as previously described. We also augment the meta-prompt for the decomposition step to include instructions describing how to suggest dependencies (see App. C).

Fig. 4b demonstrates the results of our experiment on the length reversal benchmark. We observe a transition point $5 < n^* < 7$, after which RDD outperforms CoT. Table 3 demonstrates that the time taken by RDD to complete the experiment is considerably lower than that of the CoT method.

3.4 Error analysis



(a) Error analysis for the task-specific setting.

(b) Error analysis for the generic setting.

Figure 5: The error sources of the recursive decomposition approach in the letter concatenation benchmark with respect to n_0 (the size of the list in the problem) for task-specific in-context **(a)** and generic **(b)** experiments . ϕ_d corresponds to the observed success rate in the decomposition step, ϕ_m in the merging step and ϕ_u in the unit-case. The values are computed using all problem classes c_i and within-class difficulties n_i appearing in the decomposition graph. ϕ_RDD is the end-to-end accuracy.

Error sources To analyze the sources of errors when employing RDD, we empirically quantify the accuracies ϕ_d, ϕ_u and ϕ_m of the individual steps. Using these, we estimate the overall accuracy of the system ϕ_RDD, as described in Sec. 2.1. We perform this analysis using the data from our experiments on the letter concatenation task. The results are shown in Fig. 5a for the task-specific in-context setting and Fig. 5b for the generic in-context one, with the exact numbers included in App. F. To compute these statistics, we identified the accuracies of the decomposition, unit solving, and merging steps. For decomposition and merging steps, we consider whether the problem at hand was decomposed or merged correctly. We employ an exact-match metric for all measured accuracies and average them for all instances of each task. Note that these values are averaged for all problem classes c_i and within-class difficulties n_i which appear recursively in the solving process; importantly, ϕ_u does not correspond to $\phi_\text{u}\left(c_0, n_0\right)$. In Fig. 6 (for more detail, see App. G), we provide an example of an error in the unit case which was common in our evaluation; errors made when solving a sub-problem often carry over to their parent problems during the merging step, which eventually can produce an erroneous final solution of the root problem x_0.

Error recovery Our framework is capable of recovering from the errors from the individual steps. To elicit this behavior, we include the sequence "*If you find any mistakes in the sub-solutions, you can fix the mistakes while you merge the sub-solutions*" in the merge step prompt. This is a key mechanism when modeling sub-problem dependencies: if the model does not follow the syntax to specify dependencies correctly, these may not be recognized by our parser. For instance, this may result in a sub-problem statement with missing data such as "*Reverse the following list:* ". The description-solution pair of this sub-problem will be straightforward to recognize as a mistake. Since we provide the model with the top problem description as well as the sub-problems' descriptions in the merging step, it can identify such issues and propose a merged solution correcting the erroneous sub-solutions. We have observed that, in these cases, the model simply regards the top problem as a unit case and attempts to solve it directly; if it succeeds, we deem this behavior as error recovery. In Fig. 7 (App. H), we can observe an example of the merging step in the root problem recovering from an error made when solving its sub-problems. The meaning of the merging step changes if an error recovery behavior is possible: the merging step becomes a special type of unit-solving the root problem with additional context (which may or may not be informative), and it is no longer dependent on the accuracy of unit-solving the sub-problems.

3.5 Space and time efficiency

Execution time Hypothesis 3 has also been empirically proven by our results. In the tables provided in App. E, we can observe that the time RDD takes to complete experiments is lower than the time the baselines take. Note that these values include all samples and voting calls required by self-consistency; using vanilla CoT or LtM would be faster, but we strove to compare our method to baselines with access to similar computational resources. We hypothesize that higher time efficiency is achieved due to fewer output tokens generated by our implementation. Given the quadratic space complexity of the baseline methods with respect to the difficulty of the problems, we expect that recursively decomposing the root problem will lower the number of tokens generated by the model required to reach a solution. Since each output token is generated via a full forward pass of the underlying network, it is expected that a lower amount of forward passes will result in proportional time savings. Implementing the parallelization of independent steps in RDD would further increase time savings.

Reduced context length By dividing the number of context and output tokens by the number of calls, we can see that Hypothesis 4 is confirmed: recursive decomposition helps alleviate issues relating to the overflow of the context window as the complexity of the tasks increases. We hypothesize this number is lower for RDD because of the space scaling properties of CoT-based methods.

4 Related Work

Expressive power of the reasoning graph Methods based on step-by-step decomposition, such as Chain-of-Thought (Wei et al., 2022), Socratic CoT (Shridhar et al., 2023), Least-to-Most prompting (Zhou et al., 2022), Plan-and-Solve prompting (Wang et al., 2023), iterative prompting (Wang et al., 2022a) PAL (Gao et al., 2023), Parsel (Zelikman et al., 2023) or the method proposed by Perez et al. (2020) to decompose multi-hop QA tasks, can be understood as chain-like decompositions of a problem. Tree-of-Thoughts (ToT; (Yao et al., 2023)) builds a tree; however, the structure of this graph represents a sampling process, not a recursive decomposition. Zhang et al. (2024) propose a tree-like recursive decomposition strategy, not considering sub-problem dependencies. Although DecomP (Khot et al., 2022) performs steps in sequence in a chain-like fashion, it also implicitly models the solving process as a directed acyclic graph (DAG) via tool usage, but its structure needs to be demonstrated by the user for every problem class; this graph is also not modeling dependencies between sub-problems. Graph of Thoughts (Besta et al., 2024) explicitly uses a DAG, but both its structure and the meaning of the nodes (i.e., sub-problem descriptions) must be provided by the user for every problem instance. Instead, RDD enables the model to explicitly model dependencies without user input beyond the initial problem description, resulting in a DAG.

Generic applicability The previously mentioned methods modeling the reasoning process as a chain are often able to demonstrate their decomposition strategies with generic in-context examples (e.g., few-shot CoT by Brown et al. (2020)), that is, without the user needing to provide examples for each new problem instance. However, some of these strategies cannot be applied to any arbitrary problem class, such as LtM (Libby et al., 2008; Zhou et al., 2022). More complex methods (Yao et al., 2023; Khot et al., 2022; Besta et al., 2024) 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 of problem-solving process Similar to Skeleton-of-Thought (Ning et al., 2023), we enable parallel decoding of the solution to a problem by identifying independent steps in the reasoning that can be computed in parallel. However, SoT does not decompose reasoning chains; the authors state that it is challenging to apply their method on problems that require step-by-step thinking. Other aforementioned methods relying on sequential decomposition do not allow for the parallelization of reasoning steps.

5 Conclusion

We have developed a recursive decomposition technique for LLMs allowing for sub-problem dependencies. Our empirical evaluation considered two benchmarks on six levels of increasing difficulty and two settings of varying degrees of task-specific resource availability. Based on our experiments, we also analyzed the nature of the errors our method makes during the solving process. RDD outperforms state-of-the-art baselines as the difficulty of the tasks increases. Moreover, RDD is parallelizable by design, allows for error recovery, and achieves lower time and space complexities than existing baselines.

Our results show that recursively decomposing reasoning problems with general-purpose LLMs is feasible, and can provide significant performance and resource usage benefits for complex tasks. The applicability of previous reasoning-enhancing methods to generic AI systems has been limited; the design of the RDD methodology and its demonstrated viability without task-specific support remove integration barriers towards existing LLM-based systems. We believe that our proposed method and our findings can be used to advance the reasoning capabilities of real-world language-based AI systems. Future work may explore alternative implementations of the unit-problem classifier, quantify the speedup achieved by a parallelized implementation of RDD, and develop improved strategies to elicit dependencies during the decomposition step and embed them in the merging prompt.

References

Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, et al. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805, 2023. URL https://doi.org/10.48550/arXiv.2312.11805.

Anthropic. Introducing the next generation of Claude, 2024. URL https://www.anthropic.com/news/claude-3-family.

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*, January 2024. URL https://openreview.net/forum?id=VMBWEYzmeU&referrer=%5Bthe%20profile%20of%20Robert%20Gerstenberger%5D(%2Fprofile%3Fid%3D~Robert_Gerstenberger1).

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks. *Transactions on Machine Learning Research*, June 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=YfZ4ZPt8zd.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. PaLM: Scaling Language Modeling with Pathways. *Journal of Machine Learning Research*, 24(240): 1–113, 2023. ISSN 1533-7928. URL http://jmlr.org/papers/v24/22-1144.html.

Gregoire Deletang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. Neural Networks and the Chomsky Hierarchy. In *The Eleventh International Conference on Learning Representations*, September 2022. URL https://openreview.net/forum?id=WbxHAzkeQcn.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and Fate: Limits of Transformers on Compositionality. In *Thirty-Seventh Conference on Neural Information Processing Systems*, November 2023. URL https://openreview.net/forum?id=Fkckkr3ya8.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided Language Models. In *Proceedings of the 40th International Conference on Machine Learning*, pp. 10764–10799. PMLR, July 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

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 *The Eleventh International Conference on Learning Representations*, September 2022. URL https://openreview.net/forum?id=_nGgzQjzaRy.

Myrna E Libby, Julie S Weiss, Stacie Bancroft, and William H Ahearn. A Comparison of Most-to-Least and Least-to-Most Prompting on the Acquisition of Solitary Play Skills. *Behavior Analysis in Practice*, 1(1):37–43, 2008. ISSN 1998-1929. doi: 10.1007/BF03391719. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2846579/.

Meta. Introducing Meta Llama 3: The most capable openly available LLM to date, 2024. URL https://ai.meta.com/blog/meta-llama-3/.

Suvir Mirchandani, Fei Xia, Pete Florence, Brian Ichter, Danny Driess, Montserrat Gonzalez Arenas, Kanishka Rao, Dorsa Sadigh, and Andy Zeng. Large Language Models as General Pattern Machines. In *7th Annual Conference on Robot Learning*, August 2023. URL https://openreview.net/forum?id=RcZMI8MSyE.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. Skeleton-of-Thought: Prompting LLMs for Efficient Parallel Generation. In *The Twelfth International Conference on Learning Representations*, October 2023. URL https://openreview.net/forum?id=mqVgBbNCm9.

Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the Limitations of Transformers with Simple Arithmetic Tasks, April 2021. URL http://arxiv.org/abs/2102.13019.

OpenAI. Introducing ChatGPT, November 2022. URL https://openai.com/index/chatgpt/.

OpenAI. GPT-4 Technical Report, March 2024. URL http://arxiv.org/abs/2303.08774.

Ethan Perez, Patrick Lewis, Wen-tau Yih, Kyunghyun Cho, and Douwe Kiela. Unsupervised Question Decomposition for Question Answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 8864–8880, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.713. URL https://aclanthology.org/2020.emnlp-main.713.

Alec Radford, Jeff Wu, R. Child, D. Luan, Dario Amodei, and I. Sutskever. Language Models are Unsupervised Multitask Learners. 2019. URL https://www.semanticscholar.org/paper/Language-Models-are-Unsupervised-Multitask-Learners-Radford-Wu/9405cc0d6169988371b2755e573cc28650d14dfe.

Kumar Shridhar, Alessandro Stolfo, and Mrinmaya Sachan. Distilling Reasoning Capabilities into Smaller Language Models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 7059–7073, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.441. URL https://aclanthology.org/2023.findings-acl.441.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and Efficient Foundation Language Models, February 2023. URL http://arxiv.org/abs/2302.13971.

Boshi Wang, Xiang Deng, and Huan Sun. Iteratively Prompt Pre-trained Language Models for Chain of Thought. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 2714–2730, Abu Dhabi, United Arab Emirates, December 2022a. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.174. URL https://aclanthology.org/2022.emnlp-main.174.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2609–2634, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.147. URL https://aclanthology.org/2023.acl-long.147.

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 *The Eleventh International Conference on Learning Representations*, September 2022b. URL https://openreview.net/forum?id=1PL1NIMMrw.

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 *Advances in Neural Information Processing Systems*, October 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

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 *Thirty-Seventh Conference on Neural Information Processing Systems*, November 2023. URL https://openreview.net/forum?id=5Xc1ecxO1h.

Eric Zelikman, Qian Huang, Gabriel Poesia, Noah Goodman, and Nick Haber. Parsel: Algorithmic Reasoning with Language Models by Composing Decompositions. In *Thirty-Seventh Conference on Neural Information Processing Systems*, November 2023. URL https://openreview.net/forum?id=qd9qcbVAwQ&referrer=%5Bthe%2520profile%2520of%2520Nick%2520Haber%5D(%252Fprofile%253Fid%253D~Nick_Haber1).

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, 2024. URL https://arxiv.org/abs/2402.05359.

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 *The Eleventh International Conference on Learning Representations*, September 2022. URL https://openreview.net/forum?id=WZH7099tgfM.

A Requirements for improved efficacy

We can recursively formulate the expected accuracy of our decomposition method as

$$\phi_{\text{RDD}}\left(X_0\right) = \phi_{\text{d}}\left(X_0\right)\phi_{\text{m}}\left(X_0\right)\prod_{i=1}^{w}\left[\mathbb{1}\left[\mathcal{C}\left(X_i\right)\right]\phi_{\text{u}}\left(X_i\right) + \mathbb{1}\left[\neg\mathcal{C}\left(X_i\right)\right]\phi_{\text{RDD}}\left(X_i\right)\right]. \tag{1}$$

Each random variable X_i has as domain the set of sub-problems x_i resulting from the decomposition of X_0 performed by $\mathcal{M}_\theta$, and we can attribute a class c_i and difficulty n_i to each of them. We also assume a constant width value w, corresponding to the number of sub-problems that a decomposition always produces.

We can identify several requirements for the following desideratum to hold:

$$\phi_{\text{RDD}}\left(X_0\right) > \phi_{\text{u}}\left(X_0\right). \tag{2}$$

Our desideratum states that we are more likely to arrive at a correct solution for a problem instance x_0 if we recursively decompose it instead of solving it directly as a unit case.

Theorem 1 (Decomposition and merging requirement)**.** *In order for the desideratum in Eq. (2) to hold, it is required that*

$$\phi_d\left(X_0\right)\phi_m\left(X_0\right) > \phi_u\left(X_0\right). \tag{3}$$

In other words, the probability of decomposing the problem into sub-problems and then merging their sub-solutions should be greater than the probability of solving the problem directly, so that the additional non-zero probability of obtaining wrong solutions for each of the sub-problems is balanced out. In simpler terms, the tasks of decomposing and merging a problem instance x_0 must be easier than the task of solving x_0 without decomposition.

Proof. We can prove this requirement by contradiction. Let us assume that our desideratum stated in Eq. (2) can hold when $\phi_{\text{d}}\left(X_0\right)\phi_{\text{m}}\left(X_0\right) \leq \phi_{\text{u}}\left(X_0\right)$. Without loss of generality, we can use the notation $\phi_{\text{u/RDD}}\left(X_i\right)$ to refer to the accuracies of the unit and non-unit cases for the sub-problems X_i of X_0 indifferently, $\forall i \in [1, w]$. We can employ the definition of ϕ_{RDD} given in Eq. (1) as

$$\phi_{\text{RDD}}\left(X_0\right) = \phi_{\text{d}}\left(X_0\right)\phi_{\text{m}}\left(X_0\right)\phi_{\text{u/RDD}}\left(X_1\right)\ldots\phi_{\text{u/RDD}}\left(X_w\right). \tag{4}$$

Since all accuracies are in the range $[0, 1]$, this leads to

$$\phi_{\text{RDD}}\left(X_0\right) \leq \phi_{\text{d}}\left(X_0\right)\phi_{\text{m}}\left(X_0\right). \tag{5}$$

Given our initial assumption, we can conclude that

$$\phi_{\text{RDD}}\left(X_0\right) \leq \phi_{\text{u}}\left(X_0\right), \tag{6}$$

which is a contradiction, as we stated that our desideratum would hold. □

Theorem 2 (Unit case requirement)**.** *An additional requirement for the desideratum in Eq. (2) to hold is that*

$$\phi_u\left(X_i\right) > \phi_u\left(X_0\right), \forall i \in [1, w]. \tag{7}$$

Proof. We can prove that this requirement is needed by contradiction. Assume $\phi_{\text{u}}\left(X_i\right) \leq \phi_{\text{u}}\left(X_0\right), \exists i \in [1, w]$, such that Eq. (2) holds. Let us first consider the case when $\mathcal{C}$ classifies X_i as a unit problem. All other sub-problems X_j, s.t. $j \in [1, w] \setminus \{i\}$, may be classified as either unit or non-unit problems without loss of generality. We can use Eq. (1) to formulate

$$\phi_{\text{RDD}}\left(X_0\right) = \phi_{\text{d}}\left(X_0\right)\phi_{\text{m}}\left(X_0\right)\phi_{\text{u/RDD}}\left(X_1\right)\ldots\phi_{\text{u}}\left(X_i\right)\ldots\phi_{\text{u/RDD}}\left(X_w\right). \tag{8}$$

By definition, all accuracies are in the range $[0, 1]$. Hence, we have that

$$\phi_{\text{RDD}}\left(X_0\right) \leq \phi_{\text{u}}\left(X_i\right). \tag{9}$$

Given our initial assumption, we can state that

$$\phi_{\text{RDD}}\left(X_0\right) \leq \phi_{\text{u}}\left(X_0\right), \tag{10}$$

which is a contradiction, as our initial claim was that $\phi_{\text{RDD}}\left(X_0\right) > \phi_{\text{u}}\left(X_0\right)$. We can see that this proof can be easily extended to the case where there exists more than one sub-problem X_i such that

$\phi_{\text{u}}(X_i) \leq \phi_{\text{u}}(X_0)$. Let us now consider the second case of $\mathcal{C}$ classifying X_0 as a non-unit problem. In this case, we can construct a similar proof for this recursive scenario. Similar to the previous case, we can use Eq. (1) as

$$\phi_{\text{RDD}}(X_0) = \phi_{\text{d}}(X_0)\, \phi_{\text{m}}(X_0)\, \phi_{\text{u/RDD}}(X_1) \ldots \phi_{\text{RDD}}(X_i) \ldots \phi_{\text{u/RDD}}(X_w)\,. \tag{11}$$

Since all accuracies are in the range $[0, 1]$, we have that

$$\phi_{\text{RDD}}(X_0) \leq \phi_{\text{RDD}}(X_i)\,. \tag{12}$$

If we consider X_i as the root problem of its own decomposition sub-graph, we can use the same derivation process leading to Eq. (10) to state that

$$\phi_{\text{RDD}}(X_i) \leq \phi_{\text{u}}(X_i)\,, \tag{13}$$

and thus

$$\phi_{\text{RDD}}(X_0) \leq \phi_{\text{u}}(X_i)\,. \tag{14}$$

We have reached the same statement described in Eq. (9), which we have already proven to lead to a contradiction. If we continue decomposing the sub-problems recursively, we can keep extending our proof until a unit case is reached (which can be guaranteed given strict termination criteria). $\square$

B Scheduler algorithm

The procedure for the DFS scheduler is described in Algorithm 2. The DECOMPOSE procedure corresponds to the decomposition step and the MERGEORUNIT procedure to either the unit-solving or merging steps performed by $\mathcal{M}_\theta$.

Algorithm 2 SCHEDULEDFS

```
Input: problem, visited
if problem ∈ visited then
    raise a cycle error
visited ← visited ∪ problem
if problem is decomposed then
    sub-problems ← problem.sub-problems
else
    sub-problems ← DECOMPOSE(problem)
for sub-problem ∈ sub-problems do
    SCHEDULEDFS(sub-problem, visited)
return MERGEORUNIT(problem)
```

C Prompts

Listing 1 CoT prompt, for both the baseline and unit cases.

```
Your task is to solve the problem below. You can reason about the problem before
↪  stating your answer. The answer MUST be between the following tags:
↪  <ANSWER>...</ANSWER>. An example is provided to showcase how to use the tags;
↪  you must only solve the last problem given.

## Examples

{examples}

## Problem

Problem: {problem}
Answer: Let's think step by step.
```

Listing 2 LtM prompt, for both the baseline and unit cases.

```
Your task is to solve the problem below. You can reason about the problem before
↪  stating your answer. The answer MUST be between the following tags:
↪  <ANSWER>...</ANSWER>. An example is provided to showcase how to use the tags;
↪  you must only solve the last problem given.

## Examples

{examples}

## Problem

Problem: {problem}
Answer:
```

Listing 3 RDD prompt for the decomposition step with independent sub-problems.

```
You manage {width} workers. Your task is to decompose the problem below in order
↪  to delegate sub-problems to your workers. The decomposition must be complete:
↪  combining the solutions to the sub-problems must be enough to solve the
↪  original problem. You must be brief and clear. You must consider that all
↪  sub-problems must be solved independently and that merging their solutions
↪  should produce the solution to the original problem. Do not attempt to solve
↪  the sub-problems.

If the problem is simple enough to be solved by a single worker, you must only
↪  output "This is a unit problem". Otherwise, you must propose sub-problems in a
↪  bullet list. In each bullet point, provide all necessary information for a
↪  worker to solve the sub-problem. The workers will not be provided with the
↪  original problem description nor the other sub-problems. Therefore, you must
↪  include all necessary data and instructions in the description of each
↪  sub-problem. You must only use from one up to {width} of the workers, never
↪  more than {width} workers. The sub-problems you generate can be still complex;
↪  they will be decomposed again by your workers if necessary.

You can decompose the task via either the "data decomposition strategy" or the
↪  "task decomposition strategy":

- The data decomposition strategy produces sub-problems describing exactly the
↪  same data transformation given in the original problem, applied to partitions
↪  of the input data. The partitions of the input data must be of approximately
↪  equal size. The sub-problem descriptions must be exactly the same as the
↪  description of the original problem.
- The task decomposition strategy produces sub-problems describing different data
↪  transformations, applied to exactly the same input data given in the original
↪  problem. For example, the sub-problem transformations may describe sub-steps
↪  required to solve the original problem.

Examples are provided below to illustrate some decompositions; you must only
↪  provide a decomposition for the last problem.

## Examples

{examples}

## Problem

Problem: {problem}
Answer:
```

Listing 4 RDD prompt for the merging step.

```
The problem below was decomposed into sub-problems. The sub-problems and their
↪  sub-solutions are provided in bullet points below the problem. Your task is to
↪  solve the problem with the help of the sub-solutions. Often, obtaining the
↪  final solution to the problem only requires you to apply a transformation to
↪  the sub-solutions. If you find any mistakes in the sub-solutions, you can fix
↪  the mistakes while you merge the sub-solutions.

You must reason about how to merge the sub-solutions and solve the problem before
↪  stating your final answer. The final answer MUST be between the following tags:
↪  <ANSWER>...</ANSWER>. Some examples are provided to showcase how to use the
↪  tags and to illustrate some merging strategies; you must only solve the last
↪  problem.

## Examples

{examples}

## Problem

Problem: {problem}
{subsolutions}
Answer:
```

Listing 5 RDD prompt for the decomposition step with possibly dependent sub-problems.

```
You manage {width} workers. Your task is to decompose the problem below in order to delegate sub-problems to your workers. You must only use from one up to {width} of the workers, never more than {width} workers. The decomposition must be complete: combining the solutions to the sub-problems must be enough to solve the original problem. You must be brief and clear. Do not attempt to solve the sub-problems.

If the problem is simple enough to be solved by a single worker, you must only output "This is a unit problem". Otherwise, you must propose sub-problems in a bullet list. The workers will not be provided with the original problem description nor the other sub-problem descriptions. Therefore, you must include all necessary data and instructions in the description of each sub-problem. You must never reference the original problem and you must not assume the workers can access its description and input data; instead, you must copy all relevant instructions and input data to the descriptions of sub-problems when necessary. The sub-problems you generate can be still complex; they will be decomposed again by your workers if necessary.

You can decompose the task via either the "data decomposition strategy" or the "task decomposition strategy":

- The data decomposition strategy produces sub-problems describing exactly the same data transformation given in the original problem, applied to partitions of the input data. The partitions of the input data must be of approximately equal size. The sub-problem descriptions must be exactly the same as the description of the original problem.
- The task decomposition strategy produces sub-problems describing different data transformations, applied to exactly the same input data given in the original problem. For example, the sub-problem transformations may describe sub-steps required to solve the original problem.

Each sub-problem must have a unique identifier given between square brackets before the sub-problem description. If you need to, you can also specify dependencies: within each sub-problem's description, you can refer to the solutions to other sub-problems using their identifiers between curly braces. Sub-problems cannot have the original problem as a dependency. The scheduler will substitute the identifiers of the dependencies with their solutions before sending the sub-problems to the workers. All dependencies stated between curly braces must also be sub-problems present in your bullet list.

The examples below illustrate some decompositions. You must only provide a decomposition for the last problem, do not attempt to decompose the examples.

[Continues as the prompt given in A.3.1.]
```

D In-context examples

Listing 6 CoT examples for the letter concatenation task.

```
<INPUT>Concatenate using a space the characters at index 1 of each word in the
↪  list [Gladys, Rathav, Miya]; indices start at zero.</INPUT>
<TARGET>Let's think step by step. The characters at index 1 in the input words are
↪  "l", "a" and "i". If we concatenate these, we get the answer <ANSWER>"l a
↪  i"</ANSWER>.</TARGET>

<INPUT>Concatenate using a space the characters at index 3 of each word in the
↪  list [Gloria, Ricardo, Kanwar, Chon, Manoj, Enrique, Xiong, Shaw]; indices
↪  start at zero.</INPUT>
<TARGET>Let's think step by step. The characters at index 3 in the input words are
↪  "r", "a", "w", "n", "o", "i", "n" and "w". If we concatenate these, we get the
↪  answer <ANSWER>"r a w n o i n w"</ANSWER>.</TARGET>

<INPUT>Concatenate using a space the characters at index 0 of each word in the
↪  list [Olga, Cynthia, Gladys, Cynthia, Aliyu]; indices start at zero.</INPUT>
<TARGET>Let's think step by step. The characters at index 0 in the input words are
↪  "O", "C", "G", "C" and "A". If we concatenate these, we get the answer
↪  <ANSWER>"O C G C A"</ANSWER>.</TARGET>

<INPUT>Concatenate using a space the characters at index 3 of each word in the
↪  list [Wilson]; indices start at zero.</INPUT>
<TARGET>Let's think step by step. The characters at index 3 in the input words are
↪  "s". If we concatenate these, we get the answer <ANSWER>"s"</ANSWER>.</TARGET>

<INPUT>Concatenate using a space the characters at index 2 of each word in the
↪  list [Ilya, Jacques, Francesco, Samuel, Jadhav, Rivera, Irma, Jianping, Samuel,
↪  Christian]; indices start at zero.</INPUT>
<TARGET>Let's think step by step. The characters at index 2 in the input words are
↪  "y", "c", "a", "m", "d", "v", "m", "a", "m" and "r". If we concatenate these,
↪  we get the answer <ANSWER>"y c a m d v m a m r"</ANSWER>.</TARGET>
```

Listing 7 Generic CoT examples.

```
<INPUT>Who is younger: Michael Jordan, Cristiano Ronaldo or Usain Bolt?
- Sub-problem 1: How old is Cristiano Ronaldo? Sub-solution 1: 39 years old.
- Sub-problem 2: How old is Michael Jordan? Sub-solution 2: 61 years old.
- Sub-problem 3: How old is Usain Bolt? Sub-solution 3: 37 years old.</INPUT>
<TARGET>Let's think step by step. We must compare the ages of each person:
↪  (Michael Jordan, 61) > (Cristiano Ronaldo, 39) > (Usain Bolt, 37). The answer
↪  must be the person with the lowest age. Thus, the solution is <ANSWER>Usain
↪  Bolt</ANSWER></TARGET>

<INPUT>Peter had 3 apples, 7 oranges and 12 pears. He gave 1 apple to John, 4
↪  oranges to Maria and 3 pears to Ana. How many pieces of fruit does Peter have
↪  left?</INPUT>
<TARGET>Let's think step by step. If Peter has 3 apples and gives 1 to John, he
↪  will lose 1 apple. If Peter has 7 oranges and gives 4 to Maria, he will lose 4
↪  oranges. If Peter has 12 pears and gives 3 to Ana, he will lose 3 pears. Thus,
↪  the solution is 3 - 1 + 7 - 4 + 12 - 3 = <ANSWER>14</ANSWER></TARGET>

<INPUT>What is ((((5 + 4) * 100) + 267) / (3 * 10))?</INPUT>
<TARGET>Let's think step by step. 5 + 4 = 9. 9 * 100 = 900. 900 + 267 = 1167. 3 *
↪  10 = 30. Thus, the answer is 1167 / 30 = <ANSWER>38.9</ANSWER></TARGET>

<INPUT>Which word in the list [hush, oceanic, surge, present, lie, wry, giraffe,
↪  dine, guide, urge, complete, tasteless, glorious, bird, raspy, murky, zoom,
↪  juice, select, liquid, hope, install, complete, aromatic, oceanic, fish,
↪  excited, fabricator, internal, dinosaurs, noiseless, partner] is
↪  longer?</INPUT>
<TARGET>Let's think step by step. The lengths of each word are (hush, 4), (oceanic,
↪  7), (surge, 5), (present, 7), (lie, 3), (wry, 3), (giraffe, 7), (dine, 4),
↪  (guide, 5), (urge, 4), (complete, 8), (tasteless, 9), (glorious, 8), (bird, 4),
↪  (raspy, 5), (murky, 5), (zoom, 4), (juice, 5), (select, 6), (liquid, 6), (hope,
↪  4), (install, 7), (complete, 8), (aromatic, 8), (oceanic, 7), (fish, 4),
↪  (excited, 7), (fabricator, 10), (internal, 8), (dinosaurs, 9), (noiseless, 9)
↪  and (partner, 7). Thus, the solution is <ANSWER>fabricator</ANSWER></TARGET>

<INPUT>Is the following sports-related sentence plausible? "Joao Moutinho caught
↪  the screen pass in the NFC championship."</INPUT>
<TARGET>Joao Moutinho is a soccer player. The NFC championship is part of American
↪  football, not soccer. Thus, the answer is <ANSWER>no</ANSWER></TARGET>
```

Listing 8 LtM examples for the letter concatenation task.

```
<INPUT>Concatenate using a space the characters at index 1 of each word in the
↪  list [Gladys, Rathav, Miya]; indices start at zero.</INPUT>
<TARGET>The letters at index 1 of "Gladys" and "Rathav" are "l" and "a".
↪  Concatenating "l" and "a" leads to "l a". The letter at index 1 of "Miya" is
↪  "i". Concatenating "l a" and "i" leads to <ANSWER>"l a i"</ANSWER>.</TARGET>

<INPUT>Concatenate using a space the characters at index 3 of each word in the
↪  list [Gloria, Ricardo, Kanwar, Chon, Manoj, Enrique, Xiong, Shaw]; indices
↪  start at zero.</INPUT>
<TARGET>The letters at index 3 of "Gloria" and "Ricardo" are "r" and "a".
↪  Concatenating "r" and "a" leads to "r a". The letter at index 3 of "Kanwar" is
↪  "w". Concatenating "r a" and "w" leads to "r a w". The letter at index 3 of
↪  "Chon" is "n". Concatenating "r a w" and "n" leads to "r a w n". The letter at
↪  index 3 of "Manoj" is "o". Concatenating "r a w n" and "o" leads to "r a w n
↪  o". The letter at index 3 of "Enrique" is "i". Concatenating "r a w n o" and
↪  "i" leads to "r a w n o i". The letter at index 3 of "Xiong" is "n".
↪  Concatenating "r a w n o i" and "n" leads to "r a w n o i n". The letter at
↪  index 3 of "Shaw" is "w". Concatenating "r a w n o i n" and "w" leads to
↪  <ANSWER>"r a w n o i n w"</ANSWER>.</TARGET>

<INPUT>Concatenate using a space the characters at index 0 of each word in the
↪  list [Olga, Cynthia, Gladys, Cynthia, Aliyu]; indices start at zero.</INPUT>
<TARGET>The letters at index 0 of "Olga" and "Cynthia" are "O" and "C".
↪  Concatenating "O" and "C" leads to "O C". The letter at index 0 of "Gladys" is
↪  "G". Concatenating "O C" and "G" leads to "O C G". The letter at index 0 of
↪  "Cynthia" is "C". Concatenating "O C G" and "C" leads to "O C G C". The letter
↪  at index 0 of "Aliyu" is "A". Concatenating "O C G C" and "A" leads to
↪  <ANSWER>"O C G C A"</ANSWER>.</TARGET>

<INPUT>Concatenate using a space the characters at index 3 of each word in the
↪  list [Wilson]; indices start at zero.</INPUT>
<TARGET>The letter at index 3 of "Wilson" is <ANSWER>"s"</ANSWER>.</TARGET>

<INPUT>Concatenate using a space the characters at index 2 of each word in the
↪  list [Ilya, Jacques, Francesco, Samuel, Jadhav, Rivera, Irma, Jianping, Samuel,
↪  Christian]; indices start at zero.</INPUT>
<TARGET>The letters at index 2 of "Ilya" and "Jacques" are "y" and "c".
↪  Concatenating "y" and "c" leads to "y c". The letter at index 2 of "Francesco"
↪  is "a". Concatenating "y c" and "a" leads to "y c a". The letter at index 2 of
↪  "Samuel" is "m". Concatenating "y c a" and "m" leads to "y c a m". The letter
↪  at index 2 of "Jadhav" is "d". Concatenating "y c a m" and "d" leads to "y c a
↪  m d". The letter at index 2 of "Rivera" is "v". Concatenating "y c a m d" and
↪  "v" leads to "y c a m d v". The letter at index 2 of "Irma" is "m".
↪  Concatenating "y c a m d v" and "m" leads to "y c a m d v m". The letter at
↪  index 2 of "Jianping" is "a". Concatenating "y c a m d v m" and "a" leads to
↪  "y c a m d v m a". The letter at index 2 of "Samuel" is "m". Concatenating "y
↪  c a m d v m a" and "m" leads to "y c a m d v m a m". The letter at index 2 of
↪  "Christian" is "r". Concatenating "y c a m d v m a m" and "r" leads to
↪  <ANSWER>"y c a m d v m a m r"</ANSWER>.</TARGET>
```

Listing 9 RDD examples for the decomposition step and the letter concatenation task.

```
<INPUT>Concatenate using a space the characters at index 1 of each word in the
↪  list [Dong]; indices start at zero.</INPUT>
<TARGET>This is a unit problem.</TARGET>

<INPUT>Concatenate using a space the characters at index 2 of each word in the
↪  list [Shimizu, Hoang, Muhammad, Mejia, Fernandes, Punam, Cesar]; indices start
↪  at zero.</INPUT>
<TARGET>- Concatenate using a space the characters at index 2 of each word in the
↪  list [Shimizu, Hoang, Muhammad, Mejia]; indices start at zero.
- Concatenate using a space the characters at index 2 of each word in the list
↪  [Fernandes, Punam, Cesar]; indices start at zero.</TARGET>

<INPUT>Concatenate using a space the characters at index 2 of each word in the
↪  list [Lawal, Jadhav, Sekha, Jadhav, Abraham, Sushila, Hoang, Gerhard, Heinz];
↪  indices start at zero.</INPUT>
<TARGET>- Concatenate using a space the characters at index 2 of each word in the
↪  list [Lawal, Jadhav, Sekha, Jadhav]; indices start at zero.
- Concatenate using a space the characters at index 2 of each word in the list
↪  [Abraham, Sushila, Hoang, Gerhard]; indices start at zero.
- Concatenate using a space the characters at index 2 of each word in the list
↪  [Heinz]; indices start at zero.</TARGET>

<INPUT>Concatenate using a space the characters at index 2 of each word in the
↪  list [Kailash, Ouattara, Kasongo, Perez, Jyoti]; indices start at
↪  zero.</INPUT>
<TARGET>This is a unit problem.</TARGET>

<INPUT>Concatenate using a space the characters at index 0 of each word in the
↪  list [Guan, Madina, Mejia, Herrera, Christopher, Sergey, Karina, Lucy, Ortega,
↪  Vera, Mallik, Weimin, Kwon, Zhan, Shaw, Tahir, Chang, Halyna, Weidong, Ochoa,
↪  Dung, George, Nayak, Jianming, Paola, Awad, Nabil, Garba, Amal, Sergey,
↪  Mustapha, Garcia, Bello, Sergey, Otieno, Rojas, Andrew, Mustafa, Haji, Philip,
↪  Leticia, Syed, Blanca, Mahendra, Salim, Ghulam, Quan, Yanhua, Artyom,
↪  Muhammad]; indices start at zero.</INPUT>
<TARGET>- Concatenate using a space the characters at index 0 of each word in the
↪  list [Guan, Madina, Mejia, Herrera, Christopher, Sergey, Karina, Lucy, Ortega,
↪  Vera, Mallik, Weimin]; indices start at zero.
- Concatenate using a space the characters at index 0 of each word in the list
↪  [Kwon, Zhan, Shaw, Tahir, Chang, Halyna, Weidong, Ochoa, Dung, George, Nayak,
↪  Jianming]; indices start at zero.
- Concatenate using a space the characters at index 0 of each word in the list
↪  [Paola, Awad, Nabil, Garba, Amal, Sergey, Mustapha, Garcia, Bello, Sergey,
↪  Otieno, Rojas]; indices start at zero.
- Concatenate using a space the characters at index 0 of each word in the list
↪  [Andrew, Mustafa, Haji, Philip, Leticia, Syed, Blanca, Mahendra, Salim, Ghulam,
↪  Quan, Yanhua, Artyom, Muhammad]; indices start at zero.</TARGET>
```

Listing 10 Generic RDD examples for the decomposition step.

```
<INPUT>If Peter has 3 apples and gives 1 to John, how many apples does Peter have
↪  left?</INPUT>
<TARGET>This problem is simple enough to be solved directly by a single
↪  mathematical operation. <ANSWER>This is a unit problem.</ANSWER></TARGET>

<INPUT>What is ((((5 + 4) * 100) + 267) / (3 * 10))?</INPUT>
<TARGET>We can use the data decomposition strategy here by splitting the input
↪  formula into sub-formulas. We can use two workers. The merged solution will be
↪  $sub_solution_1 / sub_solution_2$.
<ANSWER>- What is ((5 + 4) * 100) + 267?
- What is 3 * 10?</ANSWER></TARGET>

<INPUT>What is the result of log_2(16)?</INPUT>
<TARGET>This problem is simple enough to be solved directly by a single
↪  mathematical operation. <ANSWER>This is a unit problem.</ANSWER></TARGET>

<INPUT>Write the blueprint for a webpage view using the Vue3 framework about a
↪  study on salaries based on profession and age. The view must contain an
↪  initial text description of the study, a table with headers "Name", "Age",
↪  "Profession" and "Salary", as well as a picture slider. The data for the table
↪  will be available from a local JSON file, and the pictures for the slider will
↪  also be available locally.</INPUT>
<TARGET>We can use the task decomposition strategy here by splitting the task into
↪  smaller independent tasks, consisting on creating Vue3 components for each
↪  element of the view. We can use two workers. The merged solution will be the
↪  code for the components generated when solving the sub-problems, as well as
↪  code for the view using such components.
<ANSWER>- Write code using the Vue3 framework for a component representing a table
↪  with headers "Name", "Age", "Profession" and "Salary". The data for the table
↪  will be available from a local JSON file.
- Write code using the Vue3 framework for a component representing a picture
↪  slider. The pictures for the slider will be available
↪  locally.</ANSWER></TARGET>

<INPUT>Write a Python function that takes the base and height of a triangle (two
↪  floating point numbers) and returns its area (also a floating point
↪  number).</INPUT>
<TARGET>This problem is simple enough to be solved directly by writing a short
↪  Python function. <ANSWER>This is a unit problem.</ANSWER></TARGET>

<INPUT>Which word in the list [hush, oceanic, surge, present, lie, wry, giraffe,
↪  dine, guide, urge, complete, tasteless, glorious, bird, raspy, murky, zoom,
↪  juice, select, liquid, hope, install, complete, aromatic, oceanic, fish,
↪  excited, tail, internal, dinosaurs, noiseless, partner] is longer? If there is
↪  more than one word with the same length, any of them is a valid
↪  answer.</INPUT>
<TARGET>We can use the data decomposition strategy here by splitting the input
↪  list of words into smaller lists. We can use three workers. Each of the list
↪  partitions will be approximately the same size. The merged solution will be
↪  the longest word out of all the sub-solutions.
```

Listing 11 Generic RDD examples for the decomposition step (continued).

```
<ANSWER>- Which word in the list [hush, oceanic, surge, present, lie, wry, giraffe,
↪  dine, guide, urge, complete] is longer? If there is more than one word with
↪  the same length, any of them is a valid answer.
- Which word in the list [tasteless, glorious, bird, raspy, murky, zoom, juice,
↪  select, liquid, hope, install] is longer? If there is more than one word with
↪  the same length, any of them is a valid answer.
- Which word in the list [complete, aromatic, oceanic, fish, excited, tail,
↪  internal, dinosaurs, noiseless, partner] is longer? If there is more than one
↪  word with the same length, any of them is a valid answer.</ANSWER></TARGET>

<INPUT>Which word in the list [cow, banana, ensemble, castle, wise] is
↪  longer?</INPUT>
<TARGET>This problem is simple enough to be solved directly by performing a length
↪  comparison of only two words. <ANSWER>This is a unit
↪  problem.</ANSWER></TARGET>

<INPUT>Is the following sports-related sentence plausible? "Joao Moutinho caught
↪  the screen pass in the NFC championship."</INPUT>
<TARGET>We can use the task decomposition strategy here by proposing questions to
↪  gather the information required to solve the original problem. We can use two
↪  workers. The merged solution will be "yes" if the resulting information of
↪  both Joao Moutinho and the NFC championship match, and "no" otherwise.
<ANSWER>- Which sport does Joao Moutinho play?
- To which sport does the NFC championship belong to?</ANSWER></TARGET>
```

Listing 12 RDD examples for the merging step and the letter concatenation task.

```
<INPUT>Concatenate using a space the characters at index 1 of each word in the
↪  list [Orlando, Arif, Keith, Lyudmyla, Amin, Theresa, Stefan, Gilberto, Samina,
↪  Yoko, Katarzyna, Haiying, Saraswati, Theresa, Bernadette, Maung, Lopez,
↪  Pereira, Shaikh, Brown, Ortiz]; indices start at zero.
- Sub-problem 1: Concatenate using a space the characters at index 1 of each word
↪  in the list [Orlando, Arif, Keith, Lyudmyla, Amin]; indices start at zero.
↪  Sub-solution 1: "r r e y m".
- Sub-problem 2: Concatenate using a space the characters at index 1 of each word
↪  in the list [Theresa, Stefan, Gilberto, Samina, Yoko]; indices start at zero.
↪  Sub-solution 2: "h t i a o".
- Sub-problem 3: Concatenate using a space the characters at index 1 of each word
↪  in the list [Katarzyna, Haiying, Saraswati, Theresa, Bernadette]; indices
↪  start at zero. Sub-solution 3: "a a a h e".
- Sub-problem 4: Concatenate using a space the characters at index 1 of each word
↪  in the list [Maung, Lopez, Pereira, Shaikh, Brown, Ortiz]; indices start at
↪  zero. Sub-solution 4: "a o e h r r".</INPUT>
<TARGET>"r r e y m h t i a o a a a h e a o e h r r"</TARGET>

<INPUT>Concatenate using a space the characters at index 2 of each word in the
↪  list [Lawal, Jadhav, Sekha, Jadhav, Abraham, Sushila, Hoang, Gerhard, Heinz];
↪  indices start at zero.
- Sub-problem 1: Concatenate using a space the characters at index 2 of each word
↪  in the list [Lawal, Jadhav, Sekha, Jadhav]; indices start at zero.
↪  Sub-solution 1: "w d k d".
- Sub-problem 2: Concatenate using a space the characters at index 2 of each word
↪  in the list [Abraham, Sushila, Hoang, Gerhard]; indices start at zero.
↪  Sub-solution 2: "r s a r".
- Sub-problem 3: Concatenate using a space the characters at index 2 of each word
↪  in the list [Heinz]; indices start at zero. Sub-solution 3: "i".</INPUT>
<TARGET>"w d k d r s a r i"</TARGET>

<INPUT>Concatenate using a space the characters at index 1 of each word in the
↪  list [Prem, Wilson, Ashraf, Gilberto, Shobha]; indices start at zero.
- Sub-problem 1: Concatenate using a space the characters at index 1 of each word
↪  in the list [Prem, Wilson, Ashraf]; indices start at zero. Sub-solution 1: "r
↪  i s".
- Sub-problem 2: Concatenate using a space the characters at index 1 of each word
↪  in the list [Gilberto, Shobha]; indices start at zero. Sub-solution 2: "i
↪  h".</INPUT>
<TARGET>"r i s i h"</TARGET>
```

Listing 13 RDD examples for the merging step and the letter concatenation task (continued).

```
<INPUT>Concatenate using a space the characters at index 1 of each word in the
↪  list [Robin, Mostafa, Hadi, Gutierrez, Farooq, Nicolas, Alicia, Sandra,
↪  Xiaolin, Valerie]; indices start at zero.
- Sub-problem 1: Concatenate using a space the characters at index 1 of each word
↪  in the list [Robin, Mostafa, Hadi]; indices start at zero. Sub-solution 1: "o
↪  o a".
- Sub-problem 2: Concatenate using a space the characters at index 1 of each word
↪  in the list [Gutierrez, Farooq, Nicolas]; indices start at zero. Sub-solution
↪  2: "u a i".
- Sub-problem 3: Concatenate using a space the characters at index 1 of each word
↪  in the list [Alicia, Sandra, Xiaolin, Valerie]; indices start at zero.
↪  Sub-solution 3: "l a i a".</INPUT>
<TARGET>"o o a u a i l a i a"</TARGET>

<INPUT>Concatenate using a space the characters at index 1 of each word in the
↪  list [Cheng, Jianwei, Magdalena, Raimundo, Rosario, Raju, Orlando]; indices
↪  start at zero.
- Sub-problem 1: Concatenate using a space the characters at index 1 of each word
↪  in the list [Cheng, Jianwei, Magdalena]; indices start at zero. Sub-solution 1:
↪  "h i a".
- Sub-problem 2: Concatenate using a space the characters at index 1 of each word
↪  in the list [Raimundo, Rosario, Raju]; indices start at zero. Sub-solution 2:
↪  "a o a".
- Sub-problem 3: Concatenate using a space the characters at index 1 of each word
↪  in the list [Orlando]; indices start at zero. Sub-solution 3: "r".</INPUT>
<TARGET>"h i a a o a r"</TARGET>
```

Listing 14 Generic RDD examples for the merging step.

```
<INPUT>Who is younger: Michael Jordan, Cristiano Ronaldo or Usain Bolt?
- Sub-problem 1: How old is Cristiano Ronaldo? Sub-solution 1: 39 years old.
- Sub-problem 2: How old is Michael Jordan? Sub-solution 2: 61 years old.
- Sub-problem 3: How old is Usain Bolt? Sub-solution 3: 37 years old.</INPUT>
<TARGET>We can obtain the solution to the original problem by comparing the ages
↪  given in the sub-solutions. Thus, the solution is <ANSWER>Usain
↪  Bolt</ANSWER></TARGET>

<INPUT>Peter had 3 apples, 7 oranges and 12 pears. He gave 1 apple to John, 4
↪  oranges to Maria and 3 pears to Ana. How many pieces of fruit does Peter have
↪  left?
- Sub-problem 1: Peter had 3 apples and gave 1 to John. How many apples does Peter
↪  have left? Sub-solution 1: 2.
- Sub-problem 2: Peter had 7 oranges and gave 4 to Maria. How many oranges does
↪  Peter have left? Sub-solution 2: 3.
- Sub-problem 3: Peter had 12 pears and gave 3 to Ana. How many pears does Peter
↪  have left? Sub-solution 3: 9.</INPUT>
<TARGET>We can obtain the solution to the original problem by adding up the pieces
↪  Peter has left for each type of fruit. These pieces are given by each
↪  sub-solution. Thus, the solution is 2 + 3 + 9 = <ANSWER>14</ANSWER></TARGET>

<INPUT>What is ((((5 + 4) * 100) + 267) / (3 * 10))?
- Sub-problem 1: What is ((5 + 4) * 100) + 267? Sub-solution 1: 1167.
- Sub-problem 2: What is 3 * 10? Sub-solution 2: 30.</INPUT>
<TARGET>We can obtain the solution to the original problem by performing the
↪  operation $sub_solution_1 / sub_solution_2$. Thus, the solution is 1167 / 30 =
↪  <ANSWER>38.9</ANSWER></TARGET>

<INPUT>Which word in the list [hush, oceanic, surge, present, lie, wry, giraffe,
↪  dine, guide, urge, complete, tasteless, glorious, bird, raspy, murky, zoom,
↪  juice, select, liquid, hope, install, complete, aromatic, oceanic, fish,
↪  excited, fabricator, internal, dinosaurs, noiseless, partner] is longer?
- Sub-problem 1: Which word in the list [hush, oceanic, surge, present, lie, wry,
↪  giraffe, dine, guide, urge, complete] is longer? Sub-solution 1: complete.
- Sub-problem 2: Which word in the list [tasteless, glorious, bird, raspy, murky,
↪  zoom, juice, select, liquid, hope, install] is longer? Sub-solution 2:
↪  tasteless.
- Sub-problem 3: Which word in the list [complete, aromatic, oceanic, fish,
↪  excited, fabricator, internal, dinosaurs, noiseless, partner] is longer?
↪  Sub-solution 3: fabricator.</INPUT>
<TARGET>We can obtain the solution to the original problem by taking the longest
↪  word out of the three sub-solutions. "complete" has 8 letters, "tasteless" has
↪  9 letters and "fabricator" has 10 letters. Thus, the solution is
↪  <ANSWER>fabricator</ANSWER></TARGET>

<INPUT>Is the following sports-related sentence plausible? "Joao Moutinho caught
↪  the screen pass in the NFC championship."
- Sub-problem 1: Which sport does Joao Moutinho play? Sub-solution 1: Soccer.
- Sub-problem 2: To which sport does the NFC championship belong to? Sub-solution
↪  2: American football.</INPUT>
<TARGET>Joao Moutinho does not play the same sport that the NFC championship
↪  belongs to. Thus, the answer is <ANSWER>no</ANSWER></TARGET>
```

Listing 15 Generic RDD examples for the decomposition step and the length reversal task.

```
<INPUT>If Peter has 3 apples and gives 1 to John, how many apples does Peter have
↪  left?</INPUT>
<TARGET>This problem is simple enough to be solved directly by a single
↪  mathematical operation. <ANSWER>This is a unit problem.</ANSWER></TARGET>

<INPUT>Who is the brother of the Sultan of Brunei married to?</INPUT>
<TARGET>We can use the task decomposition strategy here by splitting the task into
↪  smaller tasks, in order to find out the necessary information to answer the
↪  main question. We can use two workers. The merged solution will be attained by
↪  using the intermediate information to solve the original question.
<ANSWER>- [P-1] Who is the Sultan of Brunei?
- [P-2] Who is the brother of {P-1}?
- [P-3] Who is married to {P-2}?</ANSWER></TARGET>

<INPUT>Solve for $y$: $\frac{4}{\log_2 8}x = \log_2 16 + 7$, $y = 3x$.</INPUT>
<TARGET>We can use the task decomposition strategy here by splitting the task into
↪  simpler mathematical operations. We can use three workers. The merged solution
↪  will be attained by using the intermediate results to obtain a value for $y$.
<ANSWER>- [P-1] What is the result of $\frac{4}{\log_2 8}$?
- [P-2] What is the result of $\log_2 16 + 7$?
- [P-3] What is result of $\frac{{P-2}}{{P-1}}$?</ANSWER></TARGET>

<INPUT>Write the blueprint for a webpage view using the Vue3 framework about a
↪  study on salaries based on profession and age. The view must contain an
↪  initial text description of the study, a table with headers "Name", "Age",
↪  "Profession" and "Salary", as well as a picture slider. The data for the table
↪  will be available from a local JSON file, and the pictures for the slider will
↪  also be available locally.</INPUT>
<TARGET>We can use the task decomposition strategy here by splitting the task into
↪  smaller tasks, consisting on creating Vue3 components for each element of the
↪  view. We can use two workers. The merged solution will be the code for the
↪  components generated when solving the sub-problems, as well as code for the
↪  view using such components.
<ANSWER>- [P-1] Write code using the Vue3 framework for a component representing a
↪  table with headers "Name", "Age", "Profession" and "Salary". The data for the
↪  table will be available from a local JSON file.
- [P-2] Write code using the Vue3 framework for a component representing a picture
↪  slider. The pictures for the slider will be available
↪  locally.</ANSWER></TARGET>

<INPUT>Write a Python function that takes the base and height of a triangle (two
↪  floating point numbers) and returns its area (also a floating point
↪  number).</INPUT>
<TARGET>This problem is simple enough to be solved directly by writing a short
↪  Python function. <ANSWER>This is a unit problem.</ANSWER></TARGET>
```

Listing 16 Generic RDD examples for the decomposition step and the length reversal task (continued).

```
<INPUT>Which is the oldest country out of Germany, Japan, Switzerland, Spain,
↪  Bolivia, Angola, Laos, Belgium, Canada, Mexico, Costa Rica, Indonesia,
↪  Pakistan and Rwanda?</INPUT>
<TARGET>We can use the data decomposition strategy here by splitting the input
↪  data into evenly sized partitions and solving the same problem for each
↪  partition. We can use two workers. The merged solution will be oldest country
↪  out of all the sub-solutions.
<ANSWER>- [P-1] Which is the oldest country out of Germany, Japan, Switzerland,
↪  Spain, Bolivia, Angola and Laos?
- [P-2] Which is the oldest country out of Belgium, Canada, Mexico, Costa Rica,
↪  Indonesia, Pakistan and Rwanda?</ANSWER></TARGET>

<INPUT>Which is the oldest country out of Germany, Japan, Switzerland, Spain,
↪  Bolivia, Angola and Laos?</INPUT>
<TARGET>We can use the task decomposition strategy here by performing different
↪  steps to obtain all required information to answer the question. We can use
↪  two workers. The merged solution will be the longest word out of all the
↪  sub-solutions.
<ANSWER>- [P-1] Create a list of country-age pairs for each country and their
↪  respective ages out of Germany, Japan, Switzerland, Spain, Bolivia, Angola and
↪  Laos.
- [P-2] Which is the country with the largest age, given the following list of
↪  country-age pairs: {P-1}?</ANSWER></TARGET>

<INPUT>Which word in the list [cow, banana, ensemble, castle, wise] is
↪  longer?</INPUT>
<TARGET>This problem is simple enough to be solved directly by performing a length
↪  comparison of only five words. <ANSWER>This is a unit
↪  problem.</ANSWER></TARGET>
```

E Resource usage statistics

We attempted to match the estimated resource usage of the baselines and our method by the amount of Self-Consistency (SC) (Wang et al., 2022b) samples. We used the following formula for resource matching: n_context_tokens $+ 3 \cdot$ n_output_tokens.

n_0	**Method**	**Time**	**Calls**	**Context tokens**	**Output tokens**
5	CoT+SC	2.75h	2,500	1,249,529	84,656
	LtM+SC	3.78h	1,100	1,141,800	122,996
	RDD+LtM	0.53h	506	466,812	14,572
10	CoT+SC	3.80h	2,500	1,332,860	120,837
	LtM+SC	8.87h	1,100	1,373,365	295,972
	RDD+LtM	1.20h	880	817,621	35,440
20	CoT+SC	5.82h	2,500	1,497,807	188,666
	LtM+SC	11.15h	500	889,367	378,866
	RDD+LtM	2.50h	1,541	1,416,940	75,390
50	CoT+SC	13.18h	2,700	2,166,008	437,959
	LtM+SC	17.78h*	700	1,893,349	899,997
	RDD+LtM	4.85h	2,022	2,712,931	250,710
70	CoT+SC	12,18h*	2,700	2,527,744	653,758
	LtM+SC	66.94h*	509	754,192	3,098,360
	RDD+LtM	7.10h	924	1,289,806	389,013
90	CoT+SC	25.57h*	2,700	3,536,975	1,372,692
	LtM+SC	310.12h*	421	686,331	3,121,383
	RDD+LtM	10.53h	974	1,396,950	570,232

Table 1: Resource usage for the letter concatenation benchmark with task-specific examples. Experiments were run with NVIDIA A100 GPUs; those experiments marked with an asterisk were run with NVIDIA H100 GPUs instead.

n_0	**Method**	**Time**	**Calls**	**Context tokens**	**Output tokens**
5	CoT+SC	2.75h	2,500	1,903,006	135,905
	RDD+CoT	0.57h	400	436,868	26,380
10	CoT+SC	3.93h	2,500	2,024,457	200,147
	RDD+CoT	0.63h	400	442,718	31,960
20	CoT+SC	6.38h	2,500	2,272,076	331,999
	RDD+CoT	1.83h	1,000	1,141,523	90,086
50	CoT+SC	15.03h	2,700	3,301,632	810,976
	RDD+CoT	3.48h	1,700	1,915,870	171,422
70	CoT+SC	15.09h	2,200	3,030,226	843,774
	RDD+CoT	5.32h	2,600	2,854,574	241,592
90	CoT+SC	21.30h	2,200	3,550,609	1,128,185
	RDD+CoT	5.77h	2,562	2,883,035	290,143

Table 2: Resource usage for the letter concatenation benchmark with generic examples.

n_0	**Method**	**Time**	**Calls**	**Context tokens**	**Output tokens**
3	CoT+SC	1.90h	1,500	1,238,293	97,128
	RDD+CoT	1.28h	1,012	1,095,336	62,267
5	CoT+SC	2.49h	1,500	1,641,229	127,839
	RDD+CoT	1.42h	900	1,003,953	72,215
7	CoT+SC	3.13h	1,500	1,332,821	164,456
	RDD+CoT	1.70h	902	1,015,943	86,072
10	CoT+SC	3.72h	1,500	1,742,526	194,890
	RDD+CoT	2.05h	900	1,028,783	103,765
15	CoT+SC	6.02h	1,500	1,545,019	319,041
	RDD+CoT	2.60h	900	1,053,831	135,297
20	CoT+SC	6.28h	1,500	1,945,648	335,419
	RDD+CoT	3.03h	900	1,079,213	159,455

Table 3: Resource usage for the length reversal benchmark with generic examples and RDD.

F Error analysis data

n_0	ϕ_{d}	ϕ_{m}	ϕ_{u}	ϕ_{RDD}
5	1.00	0.99	0.99	0.98
10	1.00	0.98	0.97	0.91
20	1.00	0.97	0.98	0.85
50	1.00	0.97	0.98	0.80
70	1.00	0.96	0.98	0.84
90	0.94	0.94	0.87	0.45

(a) Error analysis data for the task-specific in-context setting.

n_0	ϕ_{d}	ϕ_{m}	ϕ_{u}	ϕ_{RDD}
5	1.00	0.96	0.97	0.93
10	1.00	0.99	0.93	0.85
20	1.00	0.96	0.92	0.71
50	1.00	0.93	0.92	0.42
70	1.00	0.85	0.93	0.28
90	1.00	0.81	0.90	0.11

(b) Error analysis data for the generic in-context setting.

Table 4: A complete data account for the analysis provided in Sec. 3.4. The data for the task-specific in-context experiment is given in **(a)** and the one for the generic in-context experiment in **(b)**.

G Example of error propagation

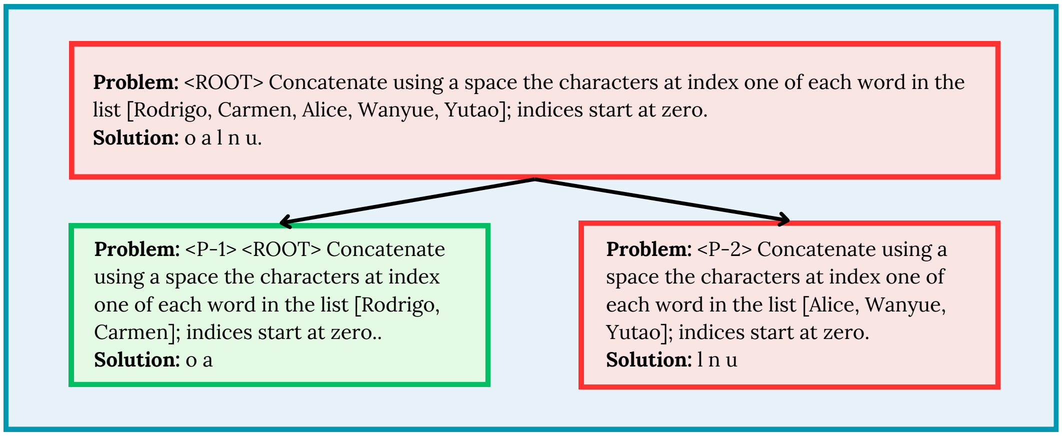


Figure 6: Example of error propagation behavior during the execution of RDD. Green nodes correspond to correctly solved problems and red nodes to incorrectly solved problems. The method performs a mistake when unit-solving *P-2*, which is carried over to the solution of the root problem.

H Example of error recovery

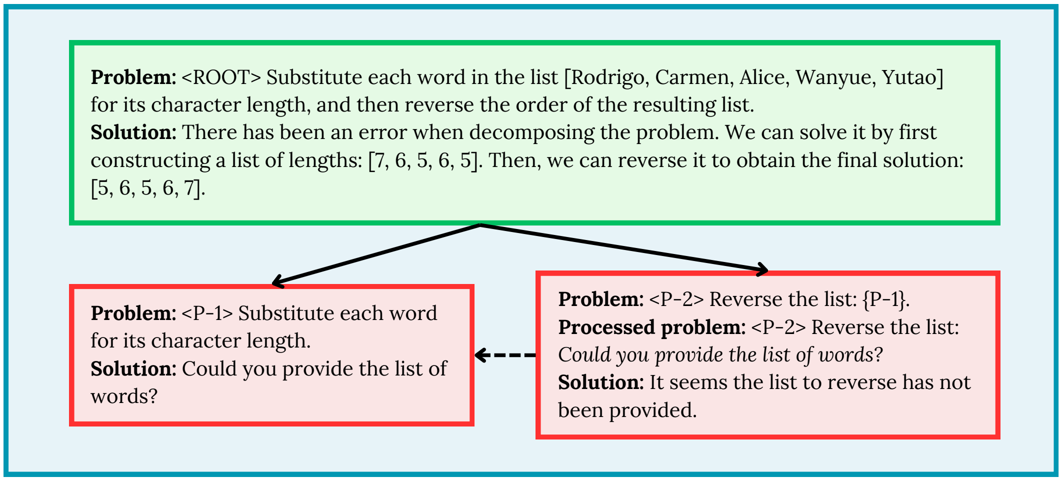
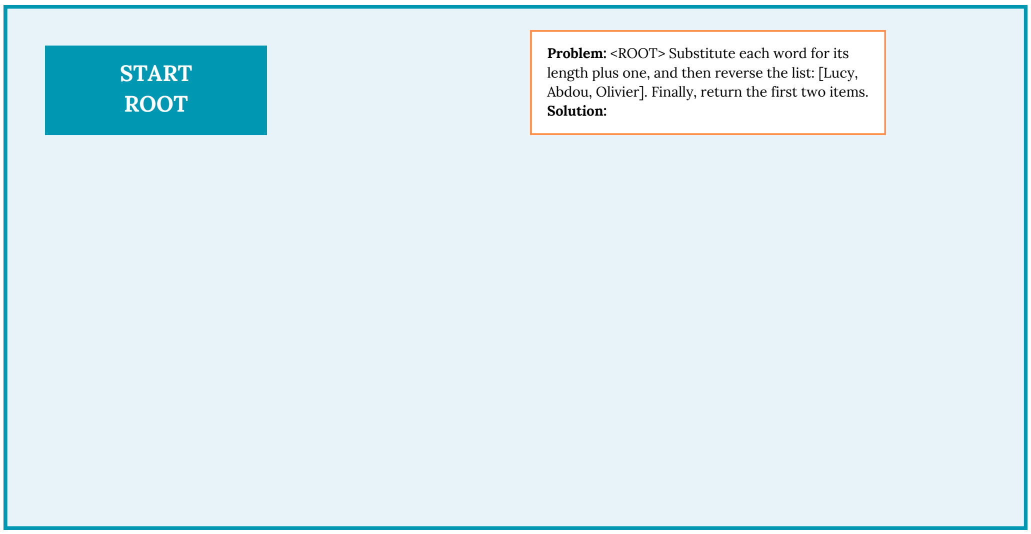
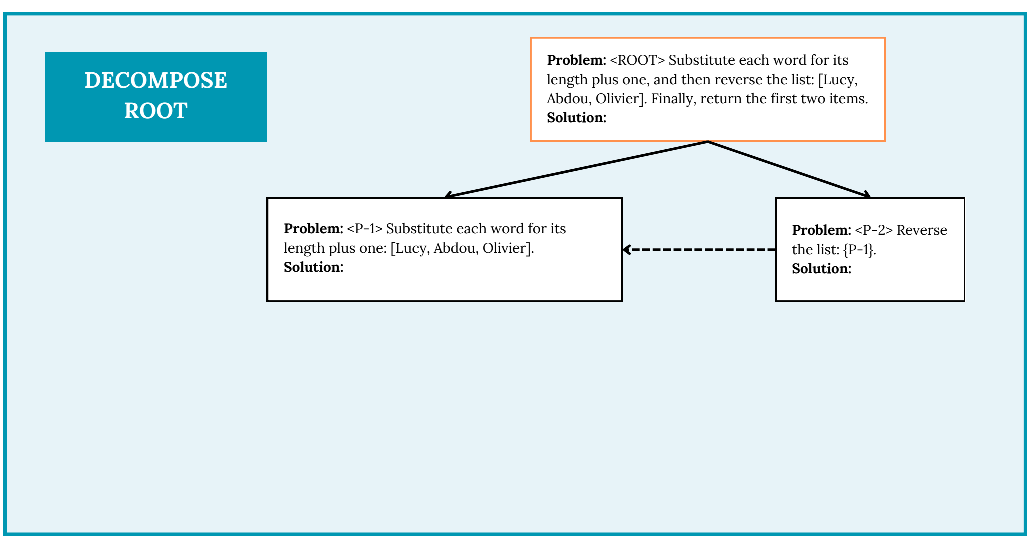


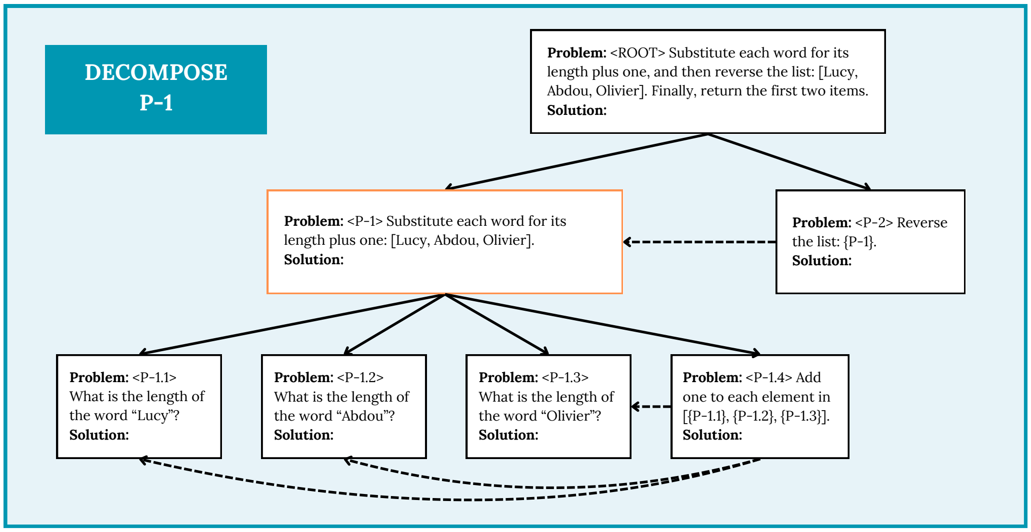
Figure 7: Example of error recovery behavior during the execution of RDD. Green nodes correspond to correctly solved problems and red nodes to incorrectly solved problems. The method does not perform the decomposition step correctly as *P-1* is formulated with missing data. This issue is carried over to *P-2*, but the merge step in the root problem recovers from this error.

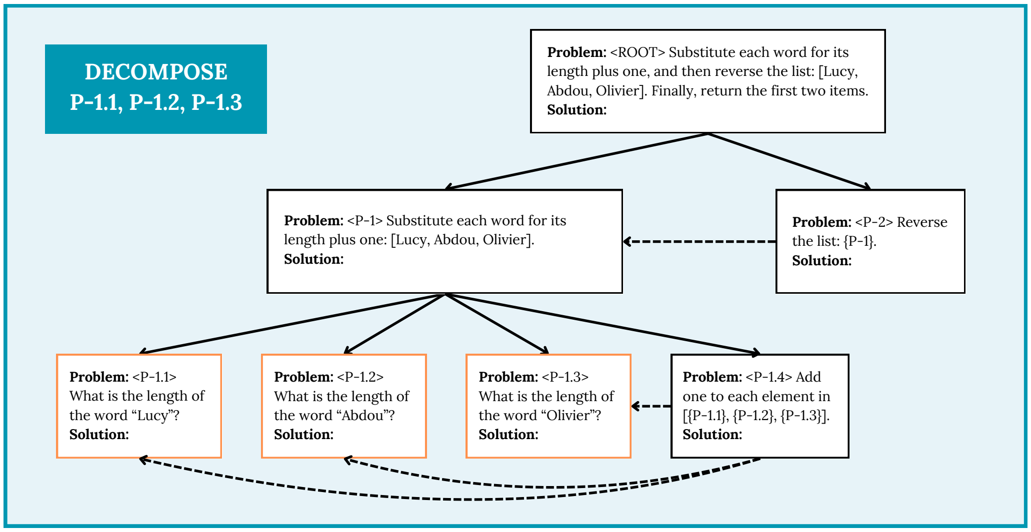
I Example execution of the RDD method

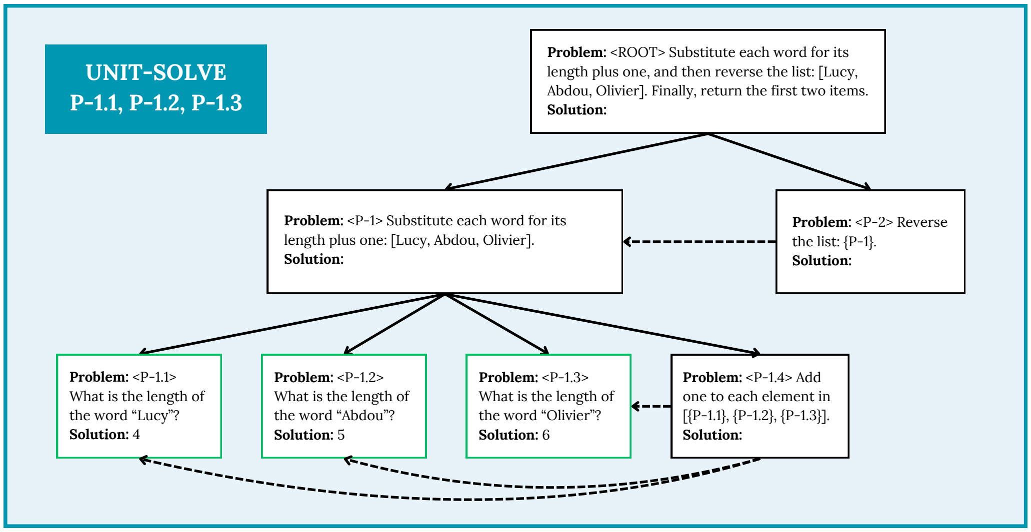
An example execution of the ScheduleBFS procedure (Algorithm 1). The execution can be followed from top to bottom. On the top-left edge of each image, we provide the type of meta-task that is performed in each step, as well as the node to which it is applied. Orange borders express a decomposition transformation of the node or an embedding of the solutions of dependencies. Green borders represent the solving process of the node, either via unit-solving or merging.

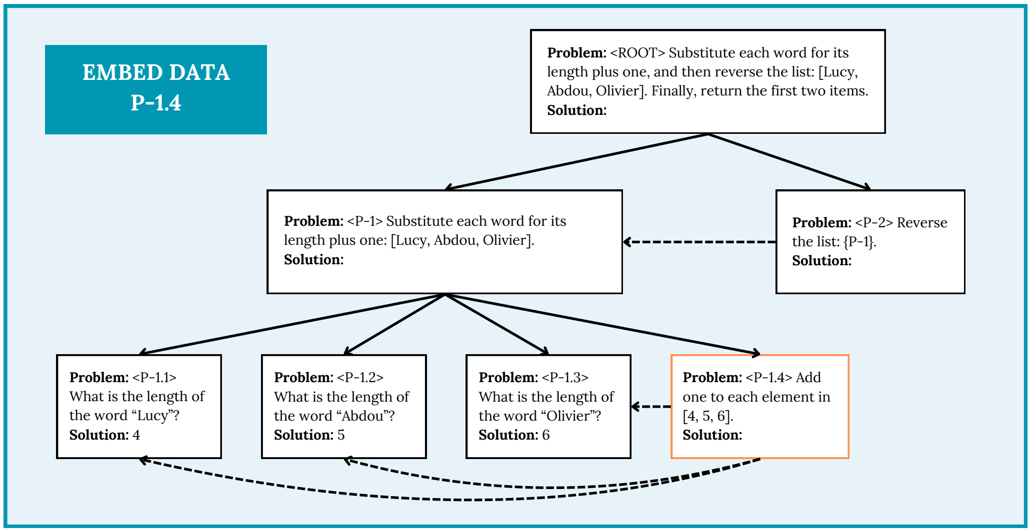


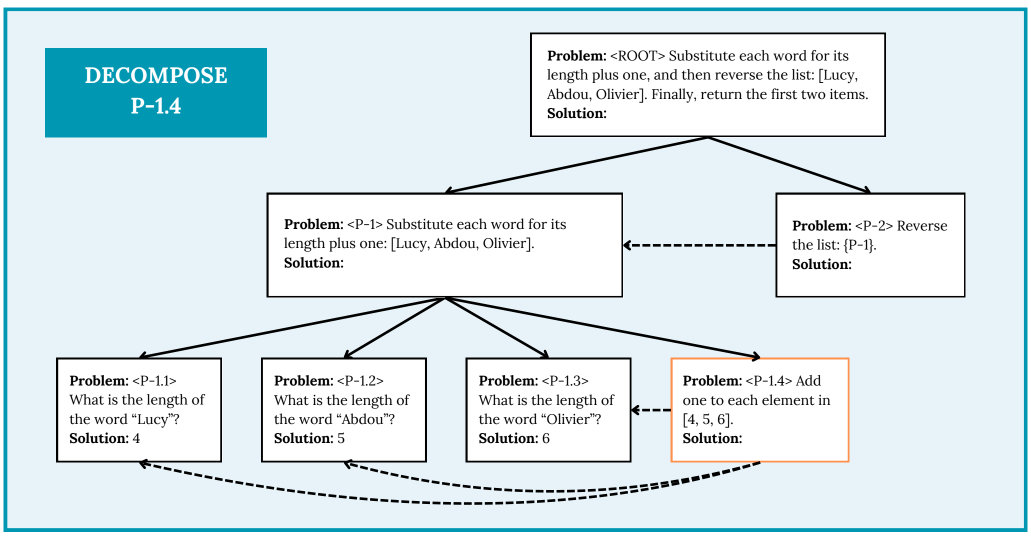


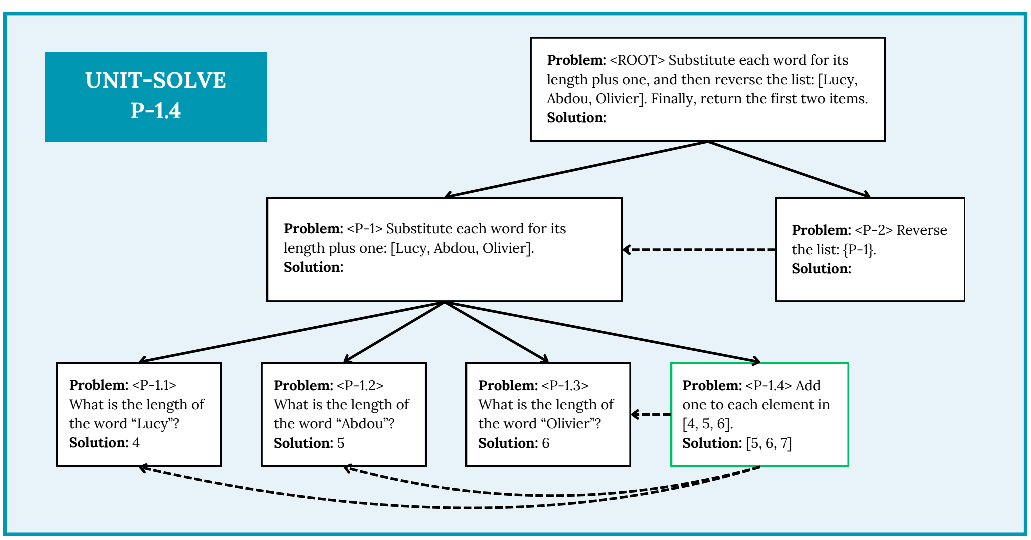

DECOMPOSE
P-1
Problem: <ROOT> Substitute each word for its length plus one, and then reverse the list: [Lucy, Abdou, Olivier]. Finally, return the first two items.
Solution:
Problem: <P-1> Substitute each word for its length plus one: [Lucy, Abdou, Olivier].
Solution:
Problem: <P-2> Reverse the list: {P-1}.
Solution:
Problem: <P-1.1> What is the length of the word "Lucy"?
Solution:
Problem: <P-1.2> What is the length of the word "Abdou"?
Solution:
Problem: <P-1.3> What is the length of the word "Olivier"?
Solution:
Problem: <P-1.4> Add one to each element in [{P-1.1}, {P-1.2}, {P-1.3}].
Solution:

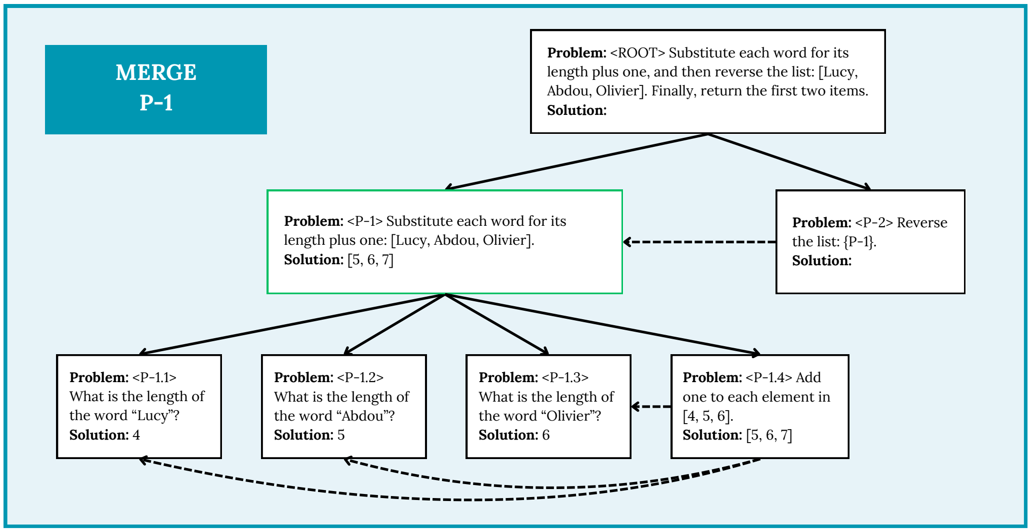

DECOMPOSE
P-1.1, P-1.2, P-1.3
Problem: <ROOT> Substitute each word for its length plus one, and then reverse the list: [Lucy, Abdou, Olivier]. Finally, return the first two items.
Solution:
Problem: <P-1> Substitute each word for its length plus one: [Lucy, Abdou, Olivier].
Solution:
Problem: <P-2> Reverse the list: {P-1}.
Solution:
Problem: <P-1.1> What is the length of the word "Lucy"?
Solution:
Problem: <P-1.2> What is the length of the word "Abdou"?
Solution:
Problem: <P-1.3> What is the length of the word "Olivier"?
Solution:
Problem: <P-1.4> Add one to each element in [{P-1.1}, {P-1.2}, {P-1.3}].
Solution:

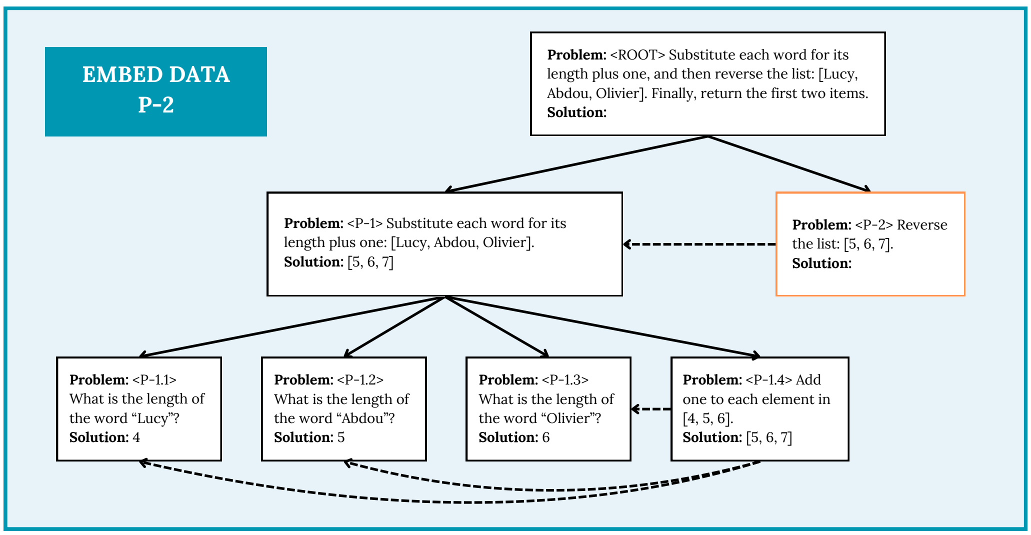

UNIT-SOLVE
P-1.1, P-1.2, P-1.3
Problem: <ROOT> Substitute each word for its length plus one, and then reverse the list: [Lucy, Abdou, Olivier]. Finally, return the first two items.
Solution:
Problem: <P-1> Substitute each word for its length plus one: [Lucy, Abdou, Olivier].
Solution:
Problem: <P-2> Reverse the list: {P-1}.
Solution:
Problem: <P-1.1> What is the length of the word "Lucy"?
Solution: 4
Problem: <P-1.2> What is the length of the word "Abdou"?
Solution: 5
Problem: <P-1.3> What is the length of the word "Olivier"?
Solution: 6
Problem: <P-1.4> Add one to each element in [{P-1.1}, {P-1.2}, {P-1.3}].
Solution:

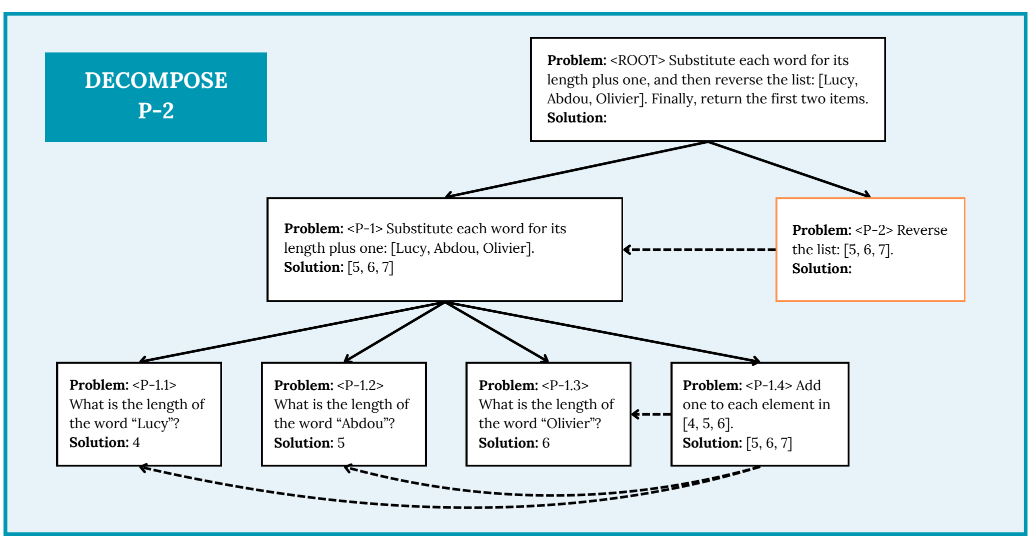

EMBED DATA
P-1.4
Problem: <ROOT> Substitute each word for its length plus one, and then reverse the list: [Lucy, Abdou, Olivier]. Finally, return the first two items.
Solution:
Problem: <P-1> Substitute each word for its length plus one: [Lucy, Abdou, Olivier].
Solution:
Problem: <P-2> Reverse the list: {P-1}.
Solution:
Problem: <P-1.1> What is the length of the word "Lucy"?
Solution: 4
Problem: <P-1.2> What is the length of the word "Abdou"?
Solution: 5
Problem: <P-1.3> What is the length of the word "Olivier"?
Solution: 6
Problem: <P-1.4> Add one to each element in [4, 5, 6].
Solution:

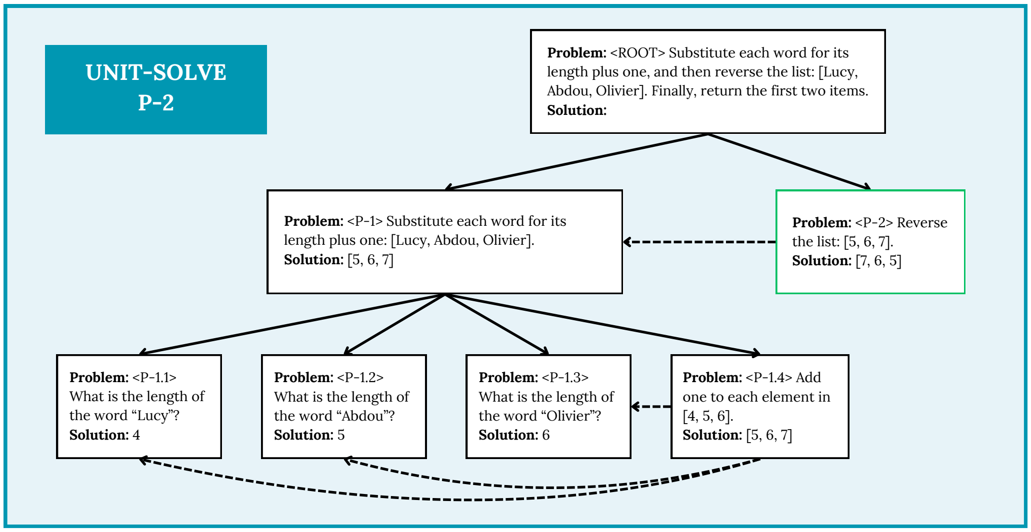

DECOMPOSE
P-1.4
Problem: <ROOT> Substitute each word for its length plus one, and then reverse the list: [Lucy, Abdou, Olivier]. Finally, return the first two items.
Solution:
Problem: <P-1> Substitute each word for its length plus one: [Lucy, Abdou, Olivier].
Solution:
Problem: <P-2> Reverse the list: {P-1}.
Solution:
Problem: <P-1.1> What is the length of the word "Lucy"?
Solution: 4
Problem: <P-1.2> What is the length of the word "Abdou"?
Solution: 5
Problem: <P-1.3> What is the length of the word "Olivier"?
Solution: 6
Problem: <P-1.4> Add one to each element in [4, 5, 6].
Solution:

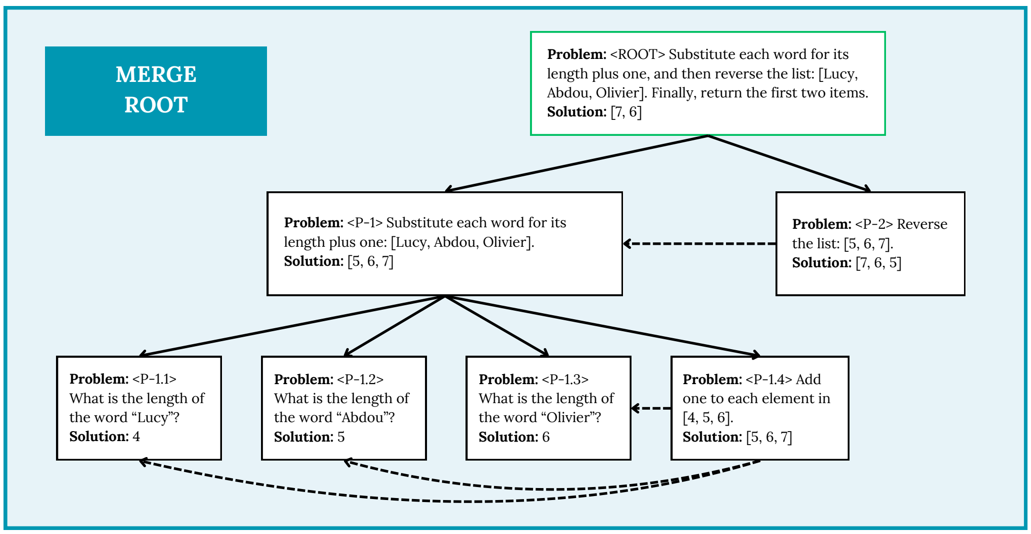

UNIT-SOLVE
P-1.4
Problem: <ROOT> Substitute each word for its length plus one, and then reverse the list: [Lucy, Abdou, Olivier]. Finally, return the first two items.
Solution:
Problem: <P-1> Substitute each word for its length plus one: [Lucy, Abdou, Olivier].
Solution:
Problem: <P-2> Reverse the list: {P-1}.
Solution:
Problem: <P-1.1> What is the length of the word "Lucy"?
Solution: 4
Problem: <P-1.2> What is the length of the word "Abdou"?
Solution: 5
Problem: <P-1.3> What is the length of the word "Olivier"?
Solution: 6
Problem: <P-1.4> Add one to each element in [4, 5, 6].
Solution: [5, 6, 7]


MERGE
P-1
Problem: <ROOT> Substitute each word for its length plus one, and then reverse the list: [Lucy, Abdou, Olivier]. Finally, return the first two items.
Solution:
Problem: <P-1> Substitute each word for its length plus one: [Lucy, Abdou, Olivier].
Solution: [5, 6, 7]
Problem: <P-2> Reverse the list: {P-1}.
Solution:
Problem: <P-1.1> What is the length of the word "Lucy"?
Solution: 4
Problem: <P-1.2> What is the length of the word "Abdou"?
Solution: 5
Problem: <P-1.3> What is the length of the word "Olivier"?
Solution: 6
Problem: <P-1.4> Add one to each element in [4, 5, 6].
Solution: [5, 6, 7]


EMBED DATA
P-2
Problem: <ROOT> Substitute each word for its length plus one, and then reverse the list: [Lucy, Abdou, Olivier]. Finally, return the first two items.
Solution:
Problem: <P-1> Substitute each word for its length plus one: [Lucy, Abdou, Olivier].
Solution: [5, 6, 7]
Problem: <P-2> Reverse the list: [5, 6, 7].
Solution:
Problem: <P-1.1> What is the length of the word "Lucy"?
Solution: 4
Problem: <P-1.2> What is the length of the word "Abdou"?
Solution: 5
Problem: <P-1.3> What is the length of the word "Olivier"?
Solution: 6
Problem: <P-1.4> Add one to each element in [4, 5, 6].
Solution: [5, 6, 7]


DECOMPOSE
P-2
Problem: <ROOT> Substitute each word for its length plus one, and then reverse the list: [Lucy, Abdou, Olivier]. Finally, return the first two items.
Solution:
Problem: <P-1> Substitute each word for its length plus one: [Lucy, Abdou, Olivier].
Solution: [5, 6, 7]
Problem: <P-2> Reverse the list: [5, 6, 7].
Solution:
Problem: <P-1.1> What is the length of the word "Lucy"?
Solution: 4
Problem: <P-1.2> What is the length of the word "Abdou"?
Solution: 5
Problem: <P-1.3> What is the length of the word "Olivier"?
Solution: 6
Problem: <P-1.4> Add one to each element in [4, 5, 6].
Solution: [5, 6, 7]


UNIT-SOLVE
P-2
Problem: <ROOT> Substitute each word for its length plus one, and then reverse the list: [Lucy, Abdou, Olivier]. Finally, return the first two items.
Solution:
Problem: <P-1> Substitute each word for its length plus one: [Lucy, Abdou, Olivier].
Solution: [5, 6, 7]
Problem: <P-2> Reverse the list: [5, 6, 7].
Solution: [7, 6, 5]
Problem: <P-1.1> What is the length of the word "Lucy"?
Solution: 4
Problem: <P-1.2> What is the length of the word "Abdou"?
Solution: 5
Problem: <P-1.3> What is the length of the word "Olivier"?
Solution: 6
Problem: <P-1.4> Add one to each element in [4, 5, 6].
Solution: [5, 6, 7]


MERGE
ROOT
Problem: <ROOT> Substitute each word for its length plus one, and then reverse the list: [Lucy, Abdou, Olivier]. Finally, return the first two items.
Solution: [7, 6]
Problem: <P-1> Substitute each word for its length plus one: [Lucy, Abdou, Olivier].
Solution: [5, 6, 7]
Problem: <P-2> Reverse the list: [5, 6, 7].
Solution: [7, 6, 5]
Problem: <P-1.1> What is the length of the word "Lucy"?
Solution: 4
Problem: <P-1.2> What is the length of the word "Abdou"?
Solution: 5
Problem: <P-1.3> What is the length of the word "Olivier"?
Solution: 6
Problem: <P-1.4> Add one to each element in [4, 5, 6].
Solution: [5, 6, 7]

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract contains a summary of the contributions and benefits of the proposed method. The introduction explicitly contains a list of the contributions and defines the problem and scope our proposed method addresses.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: In Sec. 2.1, we introduce the concept of *transition points*, before which RDD is not beneficial. In Sec. 2.2, we explain that our strategy to not provide a history of parent problems requires stronger language models to perform the decomposition step correctly. In Sec. 5, we refer to the improvements that could be made to the methodology in future work. Throughout the paper, we also mention the increased amount of computational resources our method needs to function.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: We do not present theoretical results in the main body of the paper. In Sec. 2.1, we hypothesize and formulate the appearance of *transition points*, which we later confirm empirically. We do include theoretical results in App. A; we first introduce a formulation of ϕ_{RDD} and then prove two requirements for a desideratum relevant to our work.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Yes, we provide our experimental setup in Sec. 3 and provide all prompts and in-context examples used in App. C and App. D, respectively.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We have open-sourced our implementation of the method and the code we have used to perform our experiments. We do not provide a link to it in the submission version of the paper to preserve anonymity.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The main hyperparameters and models used are mentioned in Sec. 3. The full details are available in the open-sourced code.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We include experiments with varying levels of complexity and demonstrate significance through those.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: This information is provided in App. E.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and believe we comply with it.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We address societal impact as a core contribution of the paper: we have aimed for developing a method which is generically applicable to existing intelligent systems in the real-world. Our methodology ensures that real-world users can employ our method without unfeasible data requirements.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.

- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We do not release new data or models.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We believe we have credited all creators and owners of our referenced works.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The open-source implementation of our method and the evaluation code are documented.

    Guidelines:

    - The answer NA means that the paper does not release new assets.

- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: This work does not include experiments with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This work does not include experiments with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.