Divide-or-Conquer? Which Part Should You Distill Your LLM?

Anonymous ACL submission

Abstract

Recent methods have demonstrated that Large Language Models (LLMs) can solve reasoning tasks better when they are encouraged to solve subtasks of the main task first. In this paper we devise a similar strategy that breaks down reasoning tasks into a *problem decomposition* phase and a *problem solving* phase and show that the strategy is able to outperform a single stage solution. Further, we hypothesize that the decomposition should be easier to distill into a smaller model compared to the problem solving because the latter requires large amounts of domain knowledge while the former only requires learning general problem solving strategies. We propose methods to distill these two capabilities and evaluate their impact on reasoning outcomes and inference cost. We find that we can distill the problem decomposition phase and at the same time achieve good generalization across tasks, datasets, and models. However, it is harder to distill the problem solving capability without losing performance and the resulting distilled model struggles with generalization. These results indicate that by using smaller, distilled problem decomposition models in combination with problem solving LLMs we can achieve reasoning with cost-efficient inference and local adaptation.

1 Introduction

Large Language Models (LLMs), such as GPT-4 (OpenAI, 2023), demonstrate exceptional abilities in solving knowledge-intensive tasks like Open Domain QA (ODQA) (Zhu et al., 2021), math (Yue et al., 2023), science (Taylor et al., 2022) and autonomous agents (Yao et al., 2022; Significant Gravitas, 2023; Wang et al., 2024). However, the use of gigantic LLMs with hundreds of billions of parameters can be *costly* during inference, particularly when the reasoning chain generated is lengthy. Additionally, due to the opaque nature of these black box LLMs, they offer limited *adaption* options. There is a need to use *cheaper* and more *flexible* models to leverage the power of these black box LLMs for local adaptation and *cost-efficient* inference. Distilling the large LLMs would seem like a reasonable strategy, but it often results in a significant drop in performance for downstream tasks (Chiang et al., 2023b).

Previous studies (Weng, 2023; Wang et al., 2023) have indicated that effectively addressing such tasks requires the model to proficiently perform two essential capabilities simultaneously: 1) **planning and decomposition**, which involves breaking down complex objectives into smaller, more manageable subgoals to facilitate efficient handling of intricate tasks; and 2) **execution and solving**, which involves memorizing vast amounts of knowledge from extensive web training data and effectively recalling this information when needed to execute the problem-solving process. The first capability, decomposition, typically requires the model to engage in self-reflection on the input query and generate a Chain-of-Thoughts (CoT)-style reasoning chain (Wei et al., 2022) to tackle the problem. Usually, these two abilities are intertwined in a single monolithic model throughout the problem-solving process (Zhou et al., 2022).

In this paper, we first investigate whether it is possible to decouple the decomposition and solving capabilities, and how to distill these capabilities into smaller models for faster inference. We then test several hypotheses: 1) *distilling decomposition is easier than distilling solving.* Decomposition primarily relies on semantic understanding and query parsing, while solving requires more domain expertise and knowledge. For example, decomposing the query "*who is older, Messi or Ronaldo?*" into "*how old is Messi?*", "*how old is Ronaldo*?", and "*who is older?*" only requires text comprehension, whereas solving the task necessitates memorization, retrieval, and utilization of information. We speculate that compressing the less knowledge-intensive

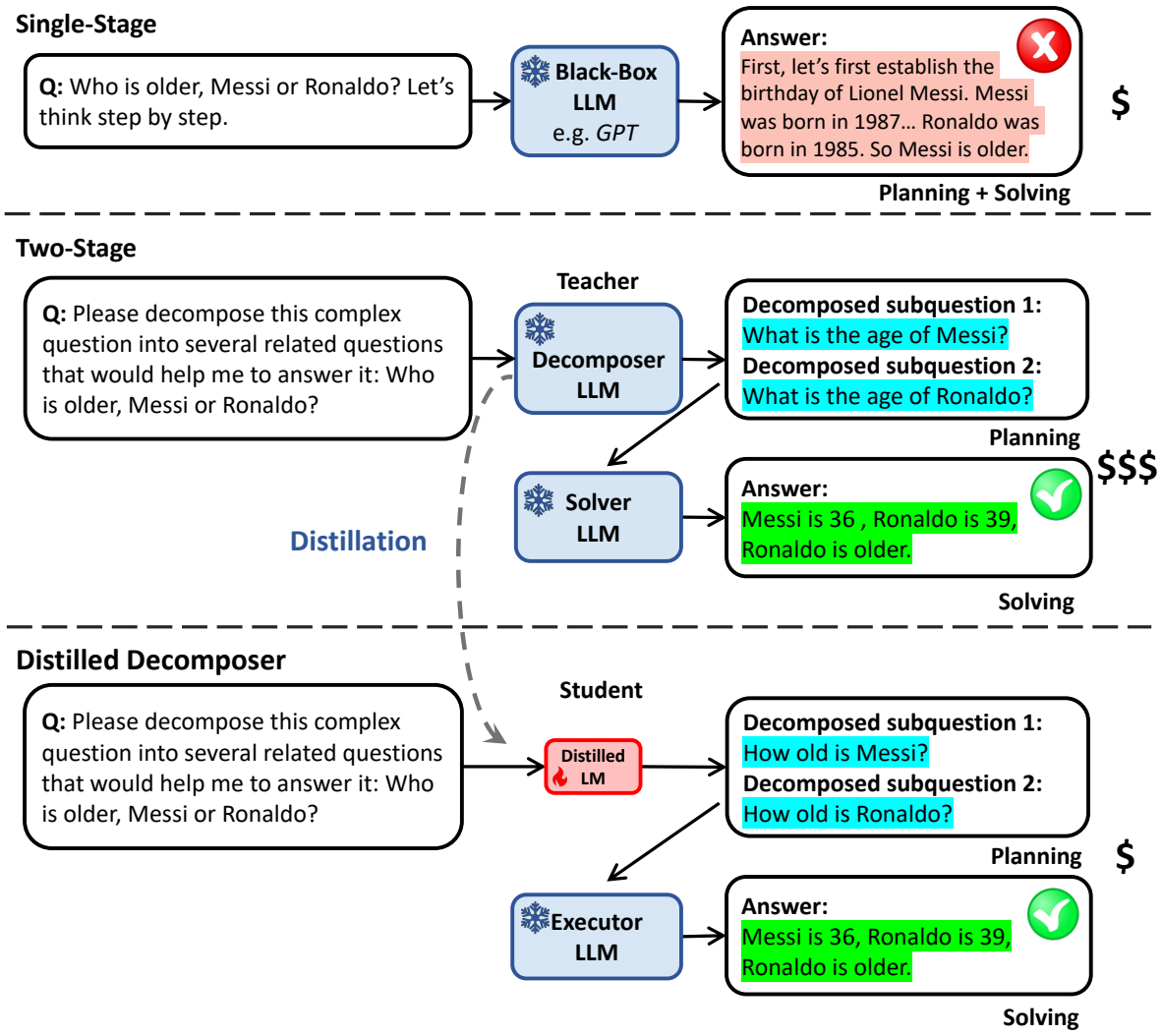

Figure 1: Reasoning with a long thought chain using the black box LLM can be expensive and inflexible. We propose to dissect the decomposition and solving of the task, and distill only the decomposition capability to a less costly and more flexible student model, while still maintaining the original performance.

decomposition is easier. 2) *decomposition is more generalizable than solving.* We hypothesize that decomposition can sometimes be abstracted into symbolic principles, making it more universally applicable across tasks, datasets, and models. This enables tasks and models to share a common decomposition engine and benefit from each other's power, reducing the effort and costs involved in distilling a model for each individual task.

A natural question arises: *is it possible to distill only the long reasoning chain, which accounts for most of the inference cost, but is relatively easier to distill?* To this end, we propose and evaluate the distillation of only the decomposition capability from the LLM. We conduct experiments using a teacher model of GPT-3.5-turbo and a student model of vicuna-13B (Chiang et al., 2023a) on QA and mathematics datasets (Dua et al., 2019; Cobbe et al., 2021). Our contributions include:

1. We demonstrate that the decomposition capability is crucial for the complex reasoning of LLM. This capability can be dissected from the problem solving or task solving capability.

2. We demonstrate the possibility and effectiveness of distilling only the query decomposition from the teacher model. The resulting distilled model can maintain most of the performance while significantly reducing inference costs. However, distilling the solving part of the LLM leads to a considerable decline in performance.

3. We show that the distilled query decomposition model exhibits good generalization across tasks, datasets, and models. However, the distilled solving for each task does not generalize

well.

2 Decoupling Decomposition and Solving

As shown in Figure 1, a common approach to solving a reasoning task using an LLM involves directly generating a response to the instruction and question. This is referred to as the **Single-Stage** model. The conventional method for LLM, known as the Chain of Thought (CoT), instructs the model to "think step by step," allowing the model to take more computational steps for difficult tasks.

However, CoT-style reasoning has limitations as it often struggles to generalize to problems beyond the scope of the in-context examples. To address this drawback, the most notable work is the Least-to-Most approach (Zhou et al., 2022), where the model breaks down the original question into subquestions and answers them sequentially. These approaches have shown improved performance compared to CoT.

For QA tasks, typically, the next subquestion is less dependent on the answer to the previous subquestions. Conveniently, we propose a *static* strategy similar to HuggingGPT (Shen et al., 2023), where in the first ***Decomposition*** stage several decomposed subquestions are first generated to decompose the primary question. In the second ***Solving*** stage, these subquestions are then answered one by one to obtain the final answer. We refer to this line of models as the **Two-Stage** models.

3 Distill the Decomposition Capability

Generating decomposed questions can be computationally expensive when the reasoning chain is long while using a black box LLM. Moreover, it is challenging to optimize or customize the decomposition process as it is performed by the black box model. Our proposal aims to address these issues by utilizing a smaller trainable student model, as a drop-in replacement for the large black box LLM for decomposition. To achieve this, we distill the decomposition capability from the teacher LLM, referred to as $\mathcal{T}$.

Generating Sub-questions from Teacher As shown in Figure 1, we begin by gathering demonstrations from $\mathcal{T}$. Instead of requesting $\mathcal{T}$ to solve the problem, we ask it to break down a given question Q without providing the solution. Specifically, we provide $\mathcal{T}$ with an instruction for decomposition, denoted as I_{decomp}, along with Q.

Instruction for decomposition: I_{decomp}

Your task is to break down a given complex question into the most relevant and helpful subquestions, ensuring that no more than three subquestions are formulated for each question. Both the context and the main question will be provided to you. If the question does not need breaking down to be answered, return "No decomposition"; otherwise, list the necessary subquestions. Only return subquestions that directly aid in answering the original question, avoiding any that could be harmful or unhelpful.
Question: Q

$\mathcal{T}$ then generates a set of sub-questions $\{S_i\}_{i=1,2,3...}$.

Decomposer Distillation Given the subquestions $\{S_i\}$ generated from the teacher, we can finetune a student decomposer $\mathcal{S}$ by optimizing the cross-entropy loss for $\mathcal{T}(I_{\text{decomp}}, Q) \rightarrow \{S_i\}$. We denote the resulting student model as $\mathcal{S}_D$-$\mathcal{T}$.

Subquestions Screening via Ground-truth Answer As an additional step, if the dataset comes with a corresponding ground-truth answer, denoted as A, we can optionally use this information to screen high-quality generated subquestions. To do this, we feed the same teacher model $\mathcal{T}$ with another instruction I_{ans} that asks the model to solve the primary question Q by first solving the subquestions $\{S_i\}$. We collect the generated answer $\mathcal{T}(I_{\text{ans}}, P, \{S_i\}, Q) \rightarrow \hat{A}$, where P represents the premise. I_{ans} is provided as the following:

Instruction for solving: I_{ans}

Solve a complex question by answering several related subquestions that would help me to answer it first. Answer the subquestions one by one and finally solve the original question. The final answer is supposed to attached in the end in the format of "The answer is: ". Now comes our primary question and its subquestions:
Premise: P
Question: Q
SubQuestion: $\{S_i\}$

We assume that, statistically speaking, good

$\{S_i\}$ will eventually lead to resolving the tasks. Thus, we can optionally filter out training instances where $\hat{A} \neq A$. However, this will result in data loss. As this screening process is similar to the *Rejection Sampling* (Touvron et al., 2023), we denote the resulting model as $\mathcal{S}_D$-R.

In Section 5.2, we compare the performance of the distilled decomposer trained using the entire set of demonstrations $\mathcal{S}_D$-$\mathcal{T}$ against decomposer trained using a screened dataset $\mathcal{S}_D$-R.

4 Experiments

Datasets We assess the effectiveness of our pipeline on two distinct datasets. GSM8K (Cobbe et al., 2021) focuses on mathematical reasoning and is composed of 7.5K training instances alongside 1K test problems. DROP (Dua et al., 2019) caters to Question Answering, containing 77.4K training samples and a 9.5K validation set. We use GSM8K test set and DROP development set for the evaluation as the DROP test set does not have oracle answer A, which limited the evaluation scenarios.

Teacher/Student Models We use GPT-3.5-Turbo-0615 model (Ouyang et al., 2022) as the teacher model throughout our experiments. After training we employ different levels of teacher models to ensure a comprehensive evaluation: one open sourced model (vanilla Vicuna (Chiang et al., 2023b)) and three black box models (text-davinci-003 (Brown et al., 2020), GPT-3.5-Turbo and GPT-4). All the student model is initialized from Vicuna-13b-v1.3 (Chiang et al., 2023a).

Student solver Models To compare the performance of distilling decomposer with distilling solver, we conducted further training on several Vicuna models to mimic the behavior of the teacher as student solvers. Similar to the student decomposer, $\mathcal{S}_E$-$\mathcal{T}$ represents the model trained using the teacher's demonstrations of $\mathcal{T}(I_{\text{ans}}, \{S_i\}, Q) \rightarrow (\{\hat{A}_i^s\}, \hat{A})$, where $\{\hat{A}_i^s\}$ represents the answers to the subquestions $\{S_i\}$ generated by $\mathcal{T}$.

Furthermore, in scenarios where the oracle answer A is available, we fine-tuned the same vanilla Vicuna-13B model to obtain $\mathcal{S}_E$-A. This model was trained using $(I_{\text{ans}}, \{S_i\}, Q) \rightarrow (\{\hat{A}_i^s\}, A)$, where the targets include answers to the subquestions $\{S_i\}$ from the $\mathcal{T}$ and the ground truth answer A.

Training Details We use a batch size of 128, train for 3 epochs on DROP and train for 5 epochs on GSM8K dataset (until convergence), and set the learning rate to $2 \cdot 10^{-5}$ for the distillation training. All the distillation fine-tuning can be finished in less than 12 hours on 8 $\times$ 80G A100 GPUs.

Inference Cost Estimation We calculate the cost based on GPT-3.5-turbo-1106 (175B), with a rate of \$0.001 for 1000 input tokens and \$0.002 for 1000 output tokens. OpenAI has made significant optimizations for inference time when serving GPT models. To ensure a fair comparison, we conservatively estimate the cost of the Vicuna-13B model by dividing the cost by the ratio of the model size. As a result, the cost for Vicuna-13B is approximately $\$7.42{*}10^{-5}$ for 1000 input tokens and $\$1.48{*}10^{-4}$ for 1000 output tokens.

5 Results

5.1 Decomposition is Essential for Reasoning

First, we explore the possibility of separating the *Decomposition* from *Solving* and assess the effectiveness of using an improved decomposition for complex reasoning tasks.

Previous studies (Press et al., 2022; Zhou et al., 2022) have demonstrated the utility of leveraging decomposed subquestions to enhance the question-answering capabilities of black-box models. They adopt *interactive* planning strategies, where the generation of each subquestion is conditioned on the answer of the previous subquestions.

As discussed in Section 2, we instead use a *static* strategy by breaking down the reasoning process into two separate stages of Decomposition and Solving. Table 1 (Single-stage GPT/Vicuna vs Two-stage GPT/Vicuna), shows that in general such a static strategy leads to performance gains over a Single-stage approach. This aligns with previous findings.

We demonstrate in Table 1 (Two-stage models) that replacing a stronger decomposer (GPT) with a weaker decomposer (Vicuna) mostly results in a noticeable decrease in performance, with an exception of using Vicuna as solver on GSM8K. We hypothesize that the reason is the Vicuna solver is too erroneous to harness the improvement from the decomposition. We observe that the decrease is more significant when the solver is more powerful. This suggests that in order to achieve optimal performance, a stronger decomposer is essential.

	Decomposer	Solver	Performance↑		Inference Expense ↓	
	Model	Model	GSM8K (EM)	DROP (F1)	GSM8K($)	DROP($)
Single-stage	-	GPT	20.32	46.51	-/0.01	-/0.05
	-	Vicuna-13B	9.40	26.68	-/0.03	-/0.03
Two-stage	GPT	GPT	65.13	55.73	0.13/0.63	0.73/0.96
	Vicuna-13B	GPT	62.93	47.13	0.02/0.67	0.07/0.96
	GPT	Vicuna-13B	28.13	21.29	0.13/0.07	0.73/0.08
	Vicuna-13B	Vicuna-13B	28.51	20.90	0.02/0.08	0.07/0.08
w/o oracle answer A	$\mathcal{S}_D$-$\mathcal{T}$	GPT	67.02	55.19	0.01/0.62	0.06/0.96
	GPT	$\mathcal{S}_E$-$\mathcal{T}$	48.98	13.37	0.13/0.09	0.73/0.06
w/ oracle answer A	$\mathcal{S}_D$-R	GPT	**67.78**	**57.97**	0.01/0.60	0.06/1.11
	GPT	$\mathcal{S}_E$-A	51.55	20.34	0.13/0.09	0.73/0.04

Table 1: Comparison results on GSM8K and DROP datasets. Performance on GSM8K is assessed via the exact match score (EM), while DROP is evaluated using the F1 score. The inference expense is estimated based on average per sample cost for each dataset. X/X indicates decomposition/solving cost.

5.2 Is Distilling Decomposition Easier than Distilling Solving?

Next, we investigate distilling knowledge from $\mathcal{T}$ to $\mathcal{S}$ when the ground truth answer A is not available. This is the most common use case as ground truth annotations are typically expensive and rare. The results are shown in Table 1 (w/o oracle answer A). It can be seen that swapping in $\mathcal{S}_D$-$\mathcal{T}$ for the decomposer is at least comparable to the performance using $\mathcal{T}$. Moreover, the $\mathcal{S}_D$-$\mathcal{T}$ exhibits a noticeable improvement compared to using Vicuna as the decomposer. However, swapping in a student solver model $\mathcal{S}_E$-$\mathcal{T}$ significantly harms the performance. We also evaluated a single-stage student model distilled from single-stage GPT. The result, omitted, was even worse than the model where GPT was the decomposer and $\mathcal{S}_E$-$\mathcal{T}$ was the solver. In terms of inference cost, our $\mathcal{S}_D$-$\mathcal{T}$ approach results in significantly lower cost for the decomposition compared to using the teacher GPT model. The cost of the solver remains relatively unchanged.

We compare some decompositions from $\mathcal{T}$, from Vicuna and from $\mathcal{S}_D$-$\mathcal{T}$ on the evaluation set in Table 2. It can be observed that the distilled $\mathcal{S}_D$-$\mathcal{T}$ model, which is obtained by using in-domain demonstration from $\mathcal{T}$, exhibits a high degree of similarity to the teacher demonstration in the generated subquestions on the unseen test set. In contrast, the original Vicuna model often generates unhelpful questions that have the potential to distract the solver.

One might naturally wonder, if a smaller student model can quickly imitate the decomposition abilities of the teacher model, why is it challenging to acquire this skill directly through student model's initial pretraining. Our hypothesis is that the decomposition ability of a stronger teacher model is easy to distill but difficult to acquire. This skill is likely based on the thorough digestion and internalization of vast amounts of data during the intensive pretraining of the larger models. However, as it is more logical and abstract rather than being *knowledge-intensive*, a few demonstrations may already provide ample guidance to the student. To draw an imperfect analogy, finding a physics theorem from massive observation is much more challenging than learning the theorem.

With available oracle answers Sometimes, we have access to the oracle answers A, which can be used to further enhance the model's performance on specific domains through local adaptation and additional finetuning. As a result, the performance on these target domain can be beyond the performance of the black-box teacher model. We explore the options to enhance the models via distillation or target domain finetuning.

In these scenarios, we can possibly use A to screen the training instance for distill the decomposer, similar to Rejection Sampling. The resulting student model $\mathcal{S}_D$-R achieved higher performance than using $\mathcal{S}_D$-$\mathcal{T}$, as shown in Table 1 (w/ oracle answer A). Notably, on the DROP dataset, $\mathcal{S}_D$-R outperforms the Teacher model in terms of F1 score.

We also finetune another Vicuna model for the solver using the ground-truth answers, referred to as $\mathcal{S}_E$-A. Our main findings remain consistent to the scenario where no oracle answers are available. Distilling the decomposer still yields better

performance comparing with finetuning the solver. We omitted the single-stage Vicuna model fine-tuned using A, which yielded worse results than GPT(decomposer) + $\mathcal{S}_E$-A(solver).

Failure modes for $\mathcal{S}_E$ models According to our observations, we hypothesize that there are two primary failure modes of the $\mathcal{S}_E$-$\mathcal{T}$ and $\mathcal{S}_E$-A models.

First, answering either subquestions or primary questions would require extensive world knowledge and commonsense, which can be difficult to compress into a student model that is hundreds of times smaller, using only a few demonstrations. In other words, a strong solving capability is knowledge-intensive. On the other hand, decomposition capability might be more compressible as it is typically more abstract, has lower information density, and is more universal than solving capability.

Second, since we used the teacher's answers to the subquestions $\{\hat{A}_i^s\}$ as part of the target, the $\mathcal{S}_E$ models could get confused and generate the final answers to one of the subquestions $\{S_i\}$, rather than the primary question Q. (Examples are provided in Appendix C.)

Based on above findings, we experimented with excluding the $\{\hat{A}_i^s\}$ in the target when training the $\mathcal{S}_E$ models. Specifically, we train the models to directly generate the answer by skipping answering subquestions, $\mathcal{S}_E(I'_{\text{ans}}, \{S_i\}, Q) \rightarrow \hat{A}/A$. The resulting models are denoted as $\mathcal{S}_E$-$\mathcal{T}$(direct) and $\mathcal{S}_E$-A(direct). We found that $\{\hat{A}_i^s\}$ from the target yields improved results over the DROP dataset, but leads to a decrease in performance over the GSM8K dataset. Overall, the decrease observed in GSM8K is more prominent than the gain seen in the DROP dataset. Therefore, we still use the $\mathcal{S}_E$ models with the $\{\hat{A}_i^s\}$ in the target. We provide additional analysis, I'_{ans}, and show the comparison results in Appendix A.

5.3 Is Distilling Decomposition More Generalizable than Distilling Solving?

Generalization to other domains We then investigate whether the distilled decomposer, which is trained on a specific domain dataset, can be applied to out-of-domain datasets with distinct objectives. To test this, we perform a cross-domain evaluation on DROP and GSM8K, which require different expertise from the solver. The results, when the oracle answer is available, are presented in Table 3. Surprisingly, the distilled decomposer $\mathcal{S}_D$-R demonstrates good generalization and versatility to the other domain, as evidenced by only a slight decrease in performance compared to using the teacher GPT model as the decomposer. In contrast, when substituting the solver with $\mathcal{S}_E$-A, which is fine-tuned on the original domain, the generalization to the other domain is poor regardless of the decomposer used. Some examples of cross-domain subquestion decomposition are shown in Table 2. The results on the scenario with no oracle answer are consistent with Table 3.

Generalization to other solvers Next, we examine whether the distilled decomposer is compatible and universally suitable for different solvers. The results can be seen in Table 4. The performance of $\mathcal{S}_D$-R is comparable to that of the teacher decomposer (GPT), and it shows overall improvements over a weaker decomposer (Vicuna) when connected to different solvers. We found that weaker solvers receive more performance gain compared to strong solvers, through upgrading to a distilled decomposer. We hypothesize that the reason lies in the fact that the weaker solver may be incapable of fully utilizing the benefits of the decomposition.

6 Ablations

We provide an extensive evaluation of various instructions, and an exploration into the influence of the number of demonstrations in Appendix B.

7 Related Work

LLM Distillation Tremendous progress (Jiao et al., 2020; Sun et al., 2019; Li et al., 2021) has been made in terms of compressing large-scale pre-trained language models such as BERT (Devlin et al., 2019) or RoBERTa (Liu et al., 2019). For generative models, compression is predominantly achieved by minimizing the K-L divergence between teacher and student distributions (Sanh et al., 2019; Gu et al., 2023). A pivotal assumption underlying these methods is the full accessibility of the teacher model's components. However, most powerful LLMs are black boxes, revealing only limited outputs. Given these constraints, several methodologies have emerged that train directly on data generated by teacher models (Chiang et al., 2023b; Taori et al., 2023). We follow a similar distillation strategy but focus on the decomposition capability distillation.

Dataset: DROP	Models	Decomposed Sub-questions
Premise P: The Raiders stayed at home for a Week 16 duel with the Houston Texans. ... The Texans tried to rally in the fourth quarter as Brown nailed a 40-yard field goal, yet the Raiders' defense would shut down any possible attempt. **Question** Q: How many field goals did both teams kick in the first half?	Vicuna-13B	1. Which teams played against each other? X 2. What were the scores for each team during the game? X 3. Which team had the lead at the end of the game? X
	GPT-3.5	1. How many field goals did the Raiders kick in the first half? 2. How many field goals did the Texans kick in the first half? 3. What is the sum of the field goals kicked by both teams in the first half?
	$\mathcal{S}_D$-$\mathcal{T}$(DROP) *In-Domain*	1. How many field goals did the Raiders kick in the first half? 2. How many field goals did the Texans kick in the first half?
	$\mathcal{S}_D$-$\mathcal{T}$(GSM) *Cross-Domain*	1. How many field goals did the Raiders kick in the first half? 2. How many field goals did the Texans kick in the first half?
Dataset: GSM8K	**Models**	**Decomposed Sub-questions**
Premise P: Mark is a copy-editor. He edits an equal number of sentences each week for two different publishers, who each pay him a different rate per sentence. Publisher B pays Mark twice what Publisher A pays. Mark edits a total number of 1000 sentences each week, and Publisher A pays him 5 cents per sentence. **Question** Q: How much does Mark make in a week, in cents?	Vicuna-13B	1. What is the rate per sentence that Publisher B pays Mark? X 2. What is the total amount Publisher A pays Mark for editing 1000 sentences? 3. What is the total amount Publisher B pays Mark for editing 1000 sentences?
	GPT-3.5	1. How many sentences does Mark edit each week for Publisher A? 2. How many sentences does Mark edit each week for Publisher B? 3. How much does Mark make per sentence from Publisher B?
	$\mathcal{S}_D$-$\mathcal{T}$(GSM) *In-Domain*	1. How many sentences does Mark edit for Publisher A in a week? 2. How many sentences does Mark edit for Publisher B in a week? 3. What is the rate per sentence paid by Publisher B?
	$\mathcal{S}_D$-$\mathcal{T}$(DROP) *Cross-Domain*	1: How much does Publisher A pay Mark per sentence? 2: How much does Publisher B pay Mark per sentence? 3: How many sentences does Mark edit in a week?

Table 2: Examples for decomposed subquestions from each method on GSM8K and DROP. $\mathcal{S}_D$-$\mathcal{T}$(GSM) and $\mathcal{S}_D$-$\mathcal{T}$(DROP) denote student models that distilled from $\mathcal{T}$'s demontration on GSM8K and DROP datasets, respectively. X indicates not helpful subquestions.

Decomposer **Solver**	GPT GPT	$\mathcal{S}_D$-R GPT	GPT $\mathcal{S}_E$-A	- $\mathcal{S}_E$-A
Trained on	**Evaluation on DROP**			
GSM8K	55.73	51.05	7.98	17.22
Trained on	**Evaluation on GSM8K**			
DROP	65.13	63.15	11.30	3.41

Table 3: Distilled student decomposers demonstrate strong generalization over out-domain datasets.

Planning and Task Decomposition of LLM-powered Agent Recent advances in LLM-powered systems have made it possible to create an end-to-end pipeline, opening up new possibilities for developing autonomous agents that can complete complex tasks using enhanced planning and memory capabilities. Promising works, such as ReAct (Yao et al., 2022), HuggingGPT (Shen et al., 2023), AutoGPT (Significant Gravitas, 2023), LangChain (Langchain-AI, 2023), GPT-Engineer (Anton Osika, 2023) and BabyAGI (Nakajima, 2023), have demonstrated significant potential in this field. These agents rely on the LLM to decompose larger tasks into more manageable components. Among them, some approaches (*e.g.*, HuggingGPT) use a *static* planning strategy by first generating the complete plan via LLM and subsequently tackling each subtask. Other approaches (*e.g.*, AutoGPT) adopt a *dynamic* and *interactive* planning strategy, where the generation of each action is conditioned on the outcome of the previous planning steps.

LLM Reasoning Chain LLMs can benefit from explicit reasoning chains, as demonstrated by recent studies (Wei et al., 2022; Zheng et al., 2023). The Chain of Thought (CoT) (Wei et al., 2022) technique has become standard for enhancing model performance on complex tasks. Tree of Thoughts (Yao et al., 2023) decomposes the problem into multiple thought steps and generates multiple thoughts per step, creating a tree structure. The LLM+P approach (Liu et al., 2023) incorporates an external classical planner for long-horizon planning and translates the plan back into natural language. Theoretical work (Feng et al., 2023) has analyzed why CoT works by using circuit complexity theory. It shows that without CoT, the model size would need to be prohibitively large to achieve the same performance through direct reasoning.

However, CoT-style reasoning is limited by the

Decomposor	Solver	GSM8K	DROP
GPT-3.5-Turbo	Vicuna-13B	28.0	33.78
	GPT-3.5-Turbo	66.0	59.38
	GPT-4	90.5	77.60
Vicuna-13B	Vicuna-13B	29.5	26.56
	GPT-3.5-Turbo	57.0	47.31
	GPT-4	88.5	79.40
$\mathcal{S}_D$-R	Vicuna-13B	31.5	33.38
	GPT-3.5-Turbo	66.5	61.94
	GPT-4	91.5	81.02

Table 4: Distilled student decomposers demonstrate consistent improvements over different solvers. Weaker solvers receive more gain.

fact that it often generalizes poorly to problems beyond the scope of the provided in-context examples (Zhou et al., 2022). To address this, some studies have asked LLMs to decompose complex questions into subquestions following the Least-to-Most prompt (Zhou et al., 2022). Others have used the self-ask method to elicit follow-up questions that aid in addressing the original inquiry (Press et al., 2022). Our work contributes to this line of research by extending the horizon to cost-efficient inference and generalization across tasks.

Question Decompostion Datasets and Approaches A widely recognized dataset for question decomposition in the literature is QDMR (Wolfson et al., 2020). It comprises an ordered list of sub-questions essential for addressing a primary question. Several previous works have been training question decomposers on the QDMR dataset (Guo et al., 2022; Zhu et al., 2023). In contrast, some research does not rely on QDMR but employs their uniquely labeled data. For instance, (Min et al., 2019) recast question decomposition as a span prediction problem and trained their model on a set of 400 labeled questions. Recognizing the challenges associated with obtaining reliable decomposition data, (Perez et al., 2020) introduced an unsupervised decomposition approach, capitalizing on the similarity between the primary question and 10M potential sub-questions mined for decomposition purposes. Our approach differs from the aforementioned methodologies because we extract the decomposition power solely from the teacher model, without relying on any annotated subquestion.

Complement LLMs with Small models There have been studies that have emphasized the potential of smaller, task-specific models to complement the predictions of LLM. Xu et al. (2023) explored a framework in which candidates produced by these task-specific models are fed to an LM, with a primary focus on classification tasks. Welleck et al. (2022) train a smaller model to iteratively improve sequences generated by LMs. Vernikos et al. (2023) have demonstrated that collecting multiple erroneous outputs from LMs and using a small corrector model to unify the generation can significantly reduce errors. Our work can also be seen as developing a smaller decomposer model to activate the best performance of a large-scale LM.

8 Conclusion

Our investigation provides a fine-grained examination of the LLM's capability on reasoning tasks, by disentangling the decomposition and solving aspects. Although both capacities are vital for reasoning, we demonstrate that decomposition is less dependent on specific knowledge and thus easier to distill compared to distilling solving capabilities, regardless of the availability of ground truth labels. Additionally, the distilled decomposer shows strong generalization abilities across different tasks, datasets and executor/solvers. For future work, it would be interesting to train universal decomposer models using data from various tasks, and explore the use of reinforcement learning to further enhance the decomposer, leveraging the signal from the solver outcome. Another possible direction for future work is to assess the effectiveness of our method in other long-horizon planning tasks, including LLM-powered agent, tool use, and multi-turn decision making.

9 Limitation

Our work is built upon several assumptions. First, we assume that the teacher model is capable of breaking down queries effectively. Second, we assume that the student model has the capacity to learn the distilled planning from the teacher model. Lastly, we assume that the tasks involved in our work require long horizon planning capability. If any of these assumptions do not hold true, it would impact the effectiveness of our proposed method.

It is important to note that we have only assessed the effectiveness of our model in the context of math and QA aspects. In order to fully complete our work, it would be necessary to evaluate our model on a broader range of planning tasks. This would include benchmarks related to tool use, LLM agents, and multiturn scenarios. Such evaluations would help verify the versatility and applicability of our proposed method.

References

Anton Osika. 2023. GPT Engineer. GitHub repository.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023a. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023b. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2368–2378, Minneapolis, Minnesota. Association for Computational Linguistics.

Guhao Feng, Yuntian Gu, Bohang Zhang, Haotian Ye, Di He, and Liwei Wang. 2023. Towards revealing the mystery behind chain of thought: a theoretical perspective. *arXiv preprint arXiv:2305.15408*.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2023. Knowledge distillation of large language models. *arXiv preprint arXiv:2306.08543*.

Xiao-Yu Guo, Yuan-Fang Li, and Gholamreza Haffari. 2022. Complex reading comprehension through question decomposition. In *Proceedings of the The 20th Annual Workshop of the Australasian Language Technology Association*, pages 31–40, Adelaide, Australia. Australasian Language Technology Association.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

Langchain-AI. 2023. Langchain Github Repository. GitHub repository.

Lei Li, Yankai Lin, Shuhuai Ren, Peng Li, Jie Zhou, and Xu Sun. 2021. Dynamic knowledge distillation for pre-trained language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Sewon Min, Victor Zhong, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2019. Multi-hop reading comprehension through question decomposition and rescoring. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6097–6109, Florence, Italy. Association for Computational Linguistics.

Yohei Nakajima. 2023. Babyagi. GitHub repository.

OpenAI. 2023. Gpt-4 technical report. *ArXiv*, abs/2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Ethan Perez, Patrick Lewis, Wen-tau Yih, Kyunghyun Cho, and Douwe Kiela. 2020. Unsupervised question decomposition for question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8864–8880, Online. Association for Computational Linguistics.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2022. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in huggingface. *arXiv preprint arXiv:2303.17580*.

Significant Gravitas. 2023. Auto-gpt: An Autonomous GPT-4 Experiment. GitHub repository.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for BERT model compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4323–4332, Hong Kong, China. Association for Computational Linguistics.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.

Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. 2022. Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Giorgos Vernikos, Arthur Bražinskas, Jakub Adamek, Jonathan Mallinson, Aliaksei Severyn, and Eric Malmi. 2023. Small language models improve giants by rewriting their outputs. *arXiv preprint arXiv:2305.13514*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2023. A survey on large language model based autonomous agents. *arXiv preprint arXiv:2308.11432*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better llm agents. *arXiv preprint arXiv:2402.01030*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. 2022. Generating sequences by learning to self-correct. *arXiv preprint arXiv:2211.00053*.

Lilian Weng. 2023. Llm powered autonomous agents. Accessed: 2024-02-13.

Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. 2020. Break it down: A question understanding benchmark. *Transactions of the Association for Computational Linguistics*, 8:183–198.

Canwen Xu, Yichong Xu, Shuohang Wang, Yang Liu, Chenguang Zhu, and Julian McAuley. 2023. Small models are valuable plug-ins for large language models. *arXiv preprint arXiv:2305.08848*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2023. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*.

Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. 2023. Progressive-hint prompting improves reasoning in large language models. *arXiv preprint arXiv:2304.09797*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

Fengbin Zhu, Wenqiang Lei, Chao Wang, Jianming Zheng, Soujanya Poria, and Tat-Seng Chua. 2021. Retrieving and reading: A comprehensive survey on open-domain question answering. *arXiv preprint arXiv:2101.00774*.

Wang Zhu, Jesse Thomason, and Robin Jia. 2023. Chain-of-questions training with latent answers for robust multistep question answering. *arXiv preprint arXiv:2305.14901*.

Appendix

A Exclusion of Answers to Subquestions

	Decomposer	Solver	Performance↑	
	Model	Model	GSM8K (EM)	DROP (F1)
w/o oracle answer A	GPT	$\mathcal{S}_E$-$\mathcal{T}$(Direct)	5.46	53.17
	GPT	$\mathcal{S}_E$-$\mathcal{T}$	48.98	13.37
w/ oracle answer A	GPT	$\mathcal{S}_E$-A(Direct)	6.44	72.55
	GPT	$\mathcal{S}_E$-A	51.55	20.34

Table 5: Excluding answers to subquestions $\{\hat{A}_i^s\}$ from the target yields improved results over the DROP dataset, but leads to a decrease in performance over the GSM8K dataset.

We hypothesize that for tasks involving mathematical reasoning, the answers typically necessitate some form of computation, making a step-by-step solution essential. Without this, setting a numerical value as the fine-tuning target almost invariably results in failure. Conversely, DROP, being a reading comprehension dataset, derives a significant portion of its answers directly from the provided text. In such scenarios, including answers to subquestions poses a risk of disrupting the answer distributions.

The instruction for solving, denoted as I'_{ans}, remain identical to those specified in I_{ans}. The only difference comes from the fine-tuning target.

B Ablation Study over Instruction for Decomposition

Decomposor	Solver	0-shot	1-shot
GPT-3.5-Turbo	GPT-3.5-Turbo	66.0	70.0
	GPT-4	90.5	91.5
Vicuna-13B	GPT-3.5-Turbo	57.0	61.5
	GPT-4	88.5	91.5
$\mathcal{S}_D$-R	GPT-3.5-Turbo	66.5	67.5
	GPT-4	91.5	91.5

Table 6: Impact of including demonstration in decomposition instruction, examined on a subset of GSM8K dataset.

Prior research has demonstrated that incorporating demonstrations within prompts can significantly enhance the ability of Large Language Models to adhere to given instructions. Our findings in Table 6 further substantiate this, revealing that including a single-shot demonstration notably improves the quality of decomposed questions. This enhancement has been consistently observed across a variety of decomposers.

Instruction	EM	f1
no restriction	45.69	56.63
no more than four	46.40	57.19
no more than three	50.00	**59.88**
no more than two	46.89	58.47

Table 7: Effect of limiting the maximum number of subquestions in decomposition instructions on a subset of the DROP dataset.

We have conducted an ablation study focusing on the instructions used for question decomposition. Our goal is for the resulting subquestions to act as useful cues for the executor, all the while ensuring they do not introduce unnecessary information. Central to our design rationale is determining the optimal number of subquestions the decomposer should produce. More specifically, we analyzed outcomes where no restrictions were applied (removing the highlighted part in I_{decomp}) and compared these against scenarios

with varying maximum numbers of subquestions allowed. The results of these investigations are detailed in Table 7. Our findings succinctly reveal that a cap of "no more than three subquestions" yields the most effective results.

Instruction for decomposition: I_{decomp}

Your task is to break down a given complex question into the most relevant and helpful subquestions, ensuring that no more than three subquestions are formulated for each question. Both the context and the main question will be provided to you. If the question does not need breaking down to be answered, return "No decomposition"; otherwise, list the necessary subquestions. Only return subquestions that directly aid in answering the original question, avoiding any that could be harmful or unhelpful.
Question: Q

C Examples Where Solver Models Become Confounded by Subquestions

As illustrated in Figure 2, up to the second subquestion, the solver model accurately responds that "The robe requires 2 bolts of blue fiber" and "it would need 1 bolt of white fiber." Nevertheless, the introduction of the third subquestion, closely resembling the second, leads to confusion. Consequently, the model deviates from its initial accuracy, culminating in an incorrect answer following this subquestion.

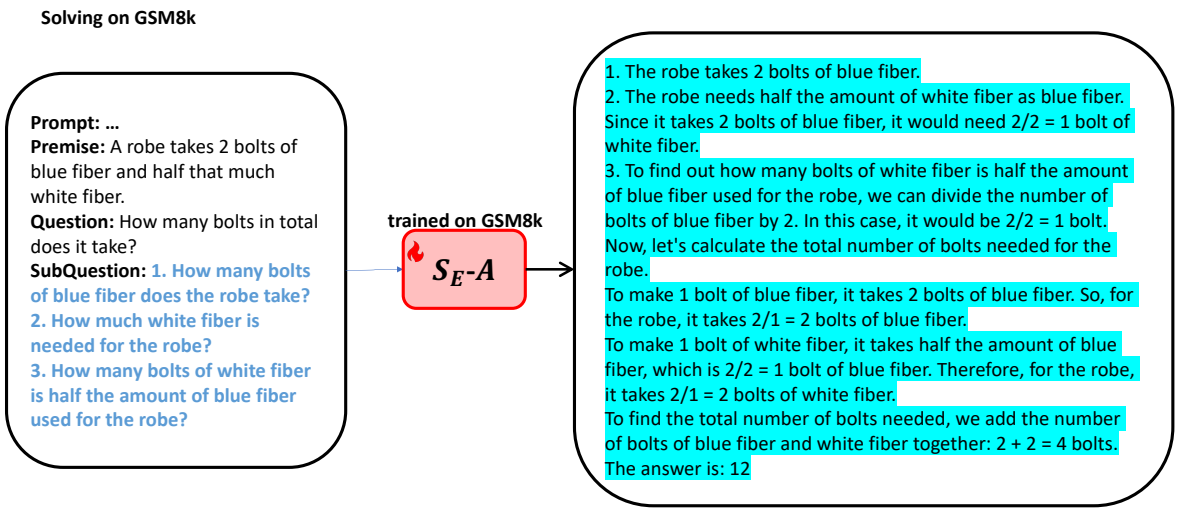

Figure 2: Solver models get lost sometimes.