

# LM<sup>2</sup>: A Simple Society of Language Models Solves Complex Reasoning

Anonymous ACL submission

## Abstract

Despite demonstrating emergent reasoning abilities, Large Language Models (LLMs) often lose track of complex, multi-step reasoning. Existing studies show that providing guidance via decomposing the original question into multiple subproblems elicits more robustness in LLM reasoning – a decomposer generates the subproblems, and a solver solves each of these subproblems. However, these techniques fail to accommodate coordination between the decomposer and the solver modules (either in a single model or different specialized ones) – the decomposer does not keep track of the ability of the solver to follow the decomposed reasoning. In this paper, we propose LM<sup>2</sup> to address these challenges. LM<sup>2</sup> modularizes the decomposition, solution, and verification into three different language models. The decomposer module identifies the key concepts necessary to solve the problem and generates step-by-step subquestions according to the reasoning requirement. The solver model generates the solution to the subproblems that are then checked by the verifier module; depending upon the feedback from the verifier, the reasoning context is constructed using the subproblems and the solutions. These models are trained to coordinate using policy learning. Exhaustive experimentation suggests the superiority of LM<sup>2</sup> over existing methods on in- and out-domain reasoning problems, outperforming the best baselines by 8.1% on MATH, 7.71% on JEEBench, and 9.7% on MedQA problems.

## 1 Introduction

Recent trends in solving complex reasoning tasks using Large Language Models (LLMs) typically follow two different dominant approaches: (i) well-curated prompting techniques (Zheng et al., 2023; Yao et al., 2024) on LLMs of exorbitant size like GPT-4 (OpenAI, 2023), or (ii) finetuning a relatively smaller LLM using domain-focused data (Shao et al., 2024; Toshniwal et al., 2024;

Dutta et al., 2024). Methods from the former category heavily rely on the proprietary LLM being used and are prone to fail absolutely when employed with less powerful models. The latter category, though cost-effective compared to humongous LLMs, often loses in generalizability due to a narrow training domain.

**The chronicle of decomposed reasoning.** A number of recent literature has pointed out that LLMs tend to perform better on complex reasoning tasks when the problem is decomposed into step-by-step subproblems (Zhou et al., 2023; Khat-tab et al., 2022; Juneja et al., 2023). Earlier techniques demonstrated the superiority by providing the model with examples containing the original problem decomposed into multiple sub-problems along with their answers (Zhou et al., 2023). However, Juneja et al. (2023) illustrated that decoupling the decomposer from the solver by finetuning a separate decomposer language model (LM) to coordinate with a larger solver LM is beneficial to simply prompting a single monolithic LM to decompose and solve. Echoing their findings, Wu et al. (2024) also found that distilling decomposition abilities from a larger LM to a smaller LM is much more generalizable compared to decomposing the solver abilities directly.

**Our contributions.** However, a major bottleneck in existing methods of decomposer finetuning is the lack of tightness between the decomposer-solver interactions. Typically, the decomposition is done in a memoryless manner, with or without the solver’s initial response; no strategy is employed to track whether the solver can follow the decomposed chain of reasoning. Towards this very end, we propose a novel multi-LLM coordination framework, **Language Model Multiplex (LM<sup>2</sup>)**. LM<sup>2</sup> is built upon three separate LMs, each dedicated to three different components of complex multistep reasoning – a **solver** LM is responsible for answering questions; a **verifier** LM provides feedback on

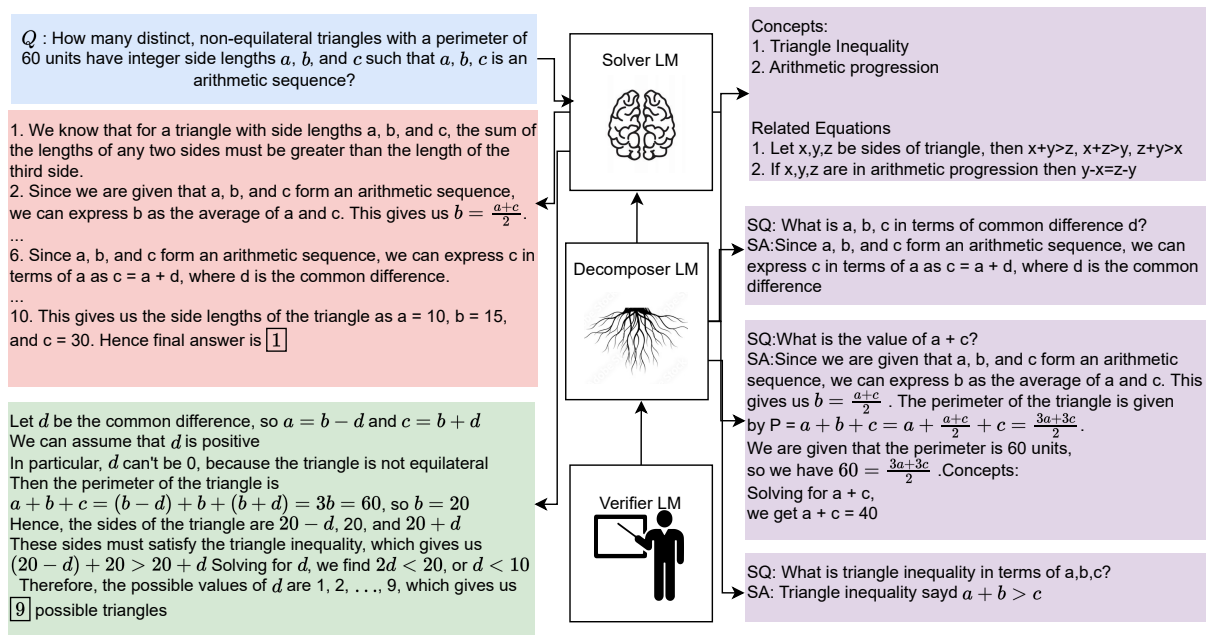


Figure 1: The inference procedure of  $LM^2$  on a question from the MATH dataset. A question (in blue) is provided to the Solver LM that produces an incorrect answer (in red). The question is then provided to the Decomposer LM that generates the concepts and step-by-step subquestions (in lilac). Each subquestion is answered by the Solver LM, and the sub-answer is verified by a Verifier LM. If the Verifier LM approves the sub-answer, that subquestion-subanswer pair is added to the context of reasoning steps; otherwise, a new subquestion is generated. The question, concepts, subquestions, and subanswers are provided in context to the Decomposer LM to generate the next subquestion. Finally, the question, concepts, subquestions, and subanswers are provided to the Solver LM to generate the final answer (in green).

084 the correctness of the output from the solver, and  
 085 a **decomposer** LM identifies the basic concepts  
 086 required to solve the problem and generates step-  
 087 by-step subproblems by decomposing the original  
 088 question (see Figure 1 for a working example). Un-  
 089 like prior approaches, the decomposer in  $LM^2$  gen-  
 090 erates each subproblem depending on the solver’s  
 091 answers to prior subproblems, along with the ver-  
 092 ifier’s feedback on those answers. Furthermore, the  
 093 decomposer generates the conceptual requirements  
 094 to solve the problem, which further streamlines  
 095 the solver LM. Irrespective of the complexity of  
 096 the underlying reasoning, the world knowledge re-  
 097 quired to answer any question is typically better  
 098 preserved in larger, proprietary LMs. Considering  
 099 this, we use GPT-3.5 (text-davinci-003) as the  
 100 solver without finetuning. For both the decomposer  
 101 and verifier, we implement parameter-efficient fine-  
 102 tuning (Hu et al., 2022) of LLaMA-2 (13 billion  
 103 parameters) separately. First, these models are fine-  
 104 tuned separately towards the tasks of decomposi-  
 105 tion and verification using datasets annotated by  
 106 GPT-4. The decomposer is then taught to coordi-  
 107 nate with the solver and the verifier models in a  
 108 policy learning setup.  $LM^2$  achieves promising per-  
 109 formance across a diverse set of reasoning tasks.

On the MATH dataset of mathematical reasoning,  
 $LM^2$  outperforms the best decomposer-tuning base-  
 line by a staggering margin 8.1% of absolute ac-  
 curacy on average. Although  $LM^2$  uses the training  
 split of the MATH dataset for tuning the decom-  
 poser and the solver, it seamlessly generalizes to  
 out-of-distribution tasks in MedQA and JEEBench,  
 outperforming the best competitive baseline with  
 9.7 % and 7.71% difference on absolute accuracy  
 respectively.

Beyond the discourse of overall numbers, we  
 perform in-depth ablation analyses to identify the  
 roles of each component of the model. We ob-  
 serve that (i) the verifier LM and concept gener-  
 ated by the decomposer LM play a crucial role in gen-  
 eralizing out-of-distribution reasoning tasks like  
 MedQA, JEEBench Chemistry, etc.; (ii) finetun-  
 ing the decomposer is crucial for better concept  
 identification – finetuned LLaMA-2 7B generates  
 more effective conceptual requirements compared  
 to even GPT-4; (iii) even while not using all the  
 modular components of  $LM^2$ , the prompt template  
 of structured reasoning boosts the performance of  
 GPT-4.<sup>1</sup>

<sup>1</sup>We are committed to release the source code and datasets upon acceptance of the paper.

## 2 Related Work

The efficacy of explicitly generating intermediate reasoning steps over direct generation of the required answer was first demonstrated by Nye et al. (2021). Chain-of-thought prompting (Wei et al., 2022) generalized the scratchpad learning of Nye et al. (2021) into an in-context learning regime using LLMs. Chain-of-thought and its successors (Chen et al., 2022; Yao et al., 2024) typically let the decomposition of a composite, multi-step reasoning problem remain implicit in the LLM.

Zhou et al. (2023) demonstrated that instead, an explicit call to the LLM to generate multiple smaller problems that are steps to answer the original query achieves more robust reasoning. Similarly, Khot et al. (2023) proposed a prompting-based problem decomposition approach where the LLM is asked to decompose a complex task using few-shot examples. However, this still burdens a single language model in handling both decomposition and solution. Juneja et al. (2023) circumvented this challenge by distilling the decomposition abilities into a relatively smaller language model. Their proposed method, DaSLaM, utilizes two separate language models that coordinate with each other to solve complex reasoning problems. Their findings suggest that finetuning the decomposer is more generalizable than finetuning the solver model. This has been further supported by Wu et al. (2024) recently. Tarasov and Shridhar (2024) explored the distillation of decomposition abilities via offline reinforcement learning. Khattab et al. (2022) proposed a programmatic retrieval augmentation framework, namely Demonstrate-Search-Predict (DSP), for knowledge-intensive generation tasks. DSP relies on the coordination between a generative LM and a retrieval model through sophisticated programs. Recent attempts have been made to incorporate dense verifiers (typically, a finetuned, bidirectional language model acting as a classifier) aiding a generative model towards robust, verifiable problem solving and text generation (Cobbe et al., 2021; Sun et al., 2023). Different techniques for verification of LM-generated outputs have been proposed subsequently, such as self-verification (Weng et al., 2023), majority voting (Li et al., 2023), etc.

## 3 Methodology

Our proposed method, LM<sup>2</sup>, is built upon the coordination of multiple LMs to perform reasoning in a modular fashion. However, such coordination

is not implicit in the pertaining stage of a model; instead, we seek to inculcate this ability via finetuning (parts of) the LM multiplex. To this end, LM<sup>2</sup> is built upon three functional components: a (preferably larger) solver model, a decomposer model, and a verifier model.

For fine-grained control over the function of the different components of LM<sup>2</sup>, we make use of a structured, step-by-step input-output framework (see Figure 1). The role of each of the modules in LM<sup>2</sup> is described as follows.

### 3.1 Decomposer

The decomposer LM guides the solver LM to solve a multi-step reasoning question in two ways. First, it provides the solver model with a set of concepts required to solve the problem. Second, it tells the solver LM what is the next sub-question required to solve given the previous sub-questions and their answers. More specifically, the decomposer LM is a function that can be defined as  $D(q, \{s_i, sa_i\}, c) : Q \times S \times SA \rightarrow \{S, C\}$ , where  $q$  represents the initial question to be solved,  $\{s_i, sa_i\}$  denotes the set of previous sub-questions ( $s_i$ ) and their corresponding answers ( $sa_i$ ), and ( $c$ ) signifies whether the function needs to predict the concept or the next sub-question.  $Q$  is the space of all the questions,  $S$  is the space of all sub-questions,  $SA$  is the space of all sub-answers, and  $C$  is the space of all concepts.

**Supervised finetuning.** The decomposer training is performed in two stages similar to (Juneja et al., 2023). The first stage is supervised finetuning, where the language model is finetuned on a dataset prepared using GPT-4. To create the dataset, we provided GPT-4 with a question and its gold reasoning. It was then asked to first generate all the concepts required to solve the question, followed by sub-questions and sub-answers. Only the questions that were answered correctly were included in the dataset. Each sample in the dataset can be expressed as a tuple  $\{Q, c, \{s_i, sa_i\}_{i=1}^n, s_{n+1}\}$ , where  $s_{n+1}$  is the next sub-question given the previous sub-questions and answers. The decomposer was then finetuned on the standard language modelling objective.

**Policy optimization.** With the supervised finetuning step, the decomposer LM is conditioned to respond to reasoning problems with concepts and decomposed subquestions. However, it is still not able to take the feedback from the solver and the verifier models into account. To this end, we uti-

lize Proximal Policy Optimization (Schulman et al., 2017) with the decomposer as the policy and the solver and the verifier model as a black-box environment. Precisely, we compute different types of rewards utilizing the feedback from the verifier model that takes the solver model’s response into account at each step and provides the decomposer with necessary refinement signals.

### 3.2 Verifier

Given the complexity of multistep reasoning, we need the verifier to be able to provide nuanced feedback to the decomposer on the possible mistakes made by the solver; a binary correct/incorrect message as employed by prior works with verifiers (Li et al., 2023; Weng et al., 2023) will limit the decomposer model’s scope of vision. For fine-grained control, the verifier is finetuned on a supervised dataset containing a question, an answer with an error made in the correct answer, a classification for the type of error, and an explanation for the classification. The verifier classifies the given input into nine classes as follows: ① Conceptual mistakes, ② Computational mistakes, ③ Procedural mistakes, ④ Misunderstood question, ⑤ Mistake in the first step, ⑥ Mistake in first half, ⑦ Mistake in second half, ⑧ Mistake in last step, and ⑨ No mistake. The dataset was produced using GPT-4, asking it to generate an explanation for the classification given the correct solution, wrong solution and the classification. The verifier is finetuned to generate the explanation and the classification (see Section 3.3 for examples of each type of error message and explanation).

### 3.3 Training with Decomposer Feedback

The training dataset curated for the decomposer LM consists of only the correct answers; hence, the decomposer is blind to the possible errors that the language model can make. In order to make the decomposer generate meaningful questions, we further finetune the decomposer while working in synergy with the solver language model using Policy gradient methods.

**Environment.** The environment consists of a black-box solver model  $\Theta$ . The model  $\Theta$  generates an answer to the current question given the concepts and previous questions and their answers.

**Policy, action and state space.** The decomposer language model  $\phi$  comprises the policy network. A state  $s$  in the state space  $S$  is defined by the concatenation of the initial state  $s_0$  and all the actions

taken from the initial state to the current state. The initial state  $s_0$  is defined as the initial question  $Q$ . The action space is defined as the token space of the language model  $\phi$ . Hence, a state  $s_n$  can be represented as  $(s_0, \{a_i\}_{i=1}^n)$ , where  $a_i$  is the action taken at the  $i_{th}$  time step.

**Reward function.** The reward is based on the feedback given by the verifier at each sub-question produced by the decomposer. The reward structure is intuitively designed to impose penalties for errors occurring in earlier sub-questions relative to those occurring in later ones. This is because fixing an early mistake can significantly increase the chances of the question being correct. Further, the policy is penalised more for conceptual and procedural mistakes as compared to computational mistakes. We construct the reward function for the  $k^{th}$  sub-question as follows:

$$R = \gamma^k \sum_{i=1}^9 r_i \quad (1)$$

where  $\gamma < 1$  is the discount factor responsible for imposing more penalties on the earlier generations.  $r_i$  are the rewards for individual feedback given by the verifier as defined below (for each type of reward, we provide an example question asked by the decomposer, an erroneous answer to that question by the solver, type of error identified and the explanation generated by the verifier in red textboxes).

*Conceptual correctness reward* is defined as,

$$r_1 = -0.15I[V(s_k, sa_k) = 1] \quad (2)$$

where  $I$  is the indicator function,  $V$  is the verifier that takes in input the  $k^{th}$  sub-question ( $s_k$ ) and its answer produced by the solver ( $sa_k$ ) and outputs the category of mistake. This reward accounts for any mistake made by the solver in understanding or while applying a concept incorrectly.

**Q:** What is  $a, b, c$  in terms of common difference  $d$ ?

**A :** Since  $a, b,$  and  $c$  form an arithmetic sequence, we can express  $c$  in terms of  $a$  as  $c = a + d$ , where  $d$  is the common difference.

**Verifier:** Conceptual Mistakes: The student misunderstood the concept of finding the possible values of the common difference in an arithmetic sequence to form a triangle with a given perimeter.

*Computational correctness reward* is defined as,

$$r_2 = -0.05I[V(s_k, sa_k) = 2] \quad (3)$$

This reward accounts for cases where the model has made any mistakes in calculation or formula



application. These kinds of mistakes are weighted less compared to all other types because an external tool can be easily used for these mechanical tasks.

**Q** : What is the value of  $a + c$ ?  
**A** : Since we are given that  $a$ ,  $b$ , and  $c$  form an arithmetic sequence, we can express  $b$  as the average of  $a$  and  $c$ . This gives us  $b = \frac{a+c}{2}$ . The perimeter of the triangle is given by  $P = a + b + c = a + \frac{a+c}{2} + c = \frac{3a+3c}{2}$ . We are given that the perimeter is 60 units, so we have  $60 = \frac{3a+3c}{2}$ . Solving for  $a + c$ , we get  $a + c = 20$   
**Verifier** : Calculation Mistakes: The model has made a calculation mistake while calculating the value of  $a + c$ .  $a + c$  should be 40.

*Procedural correctness reward* is defined as,

$$r_3 = -0.15I[V(s_k, sa_k) = 3] \quad (4)$$

This encompasses any mistake made by the model while following a step-by-step procedure whose information is already given in-context while solving the problem.

**Q** : What are the possible values of  $a, d$ ?  
**A** : Substituting this into the equation  $a + c = 20$ , we get  $a + (a + d) = 20$ , which simplifies to  $2a + d = 20 - a$ . Since the side lengths must be integers,  $a$  and  $d$  must be integers. We can then try different values of  $a$  and  $d$  that satisfy the equation  $2a + d = 20 - a$ . After trying different values, we find that the only solution that gives integer side lengths is  $a = 10$  and  $d = 20$ .  
**Verifier** : Procedural Mistakes: The student made a mistake in the procedure of solving the problem by not accurately finding the possible values of the common difference.

*Question understanding reward* is defined as,

$$r_4 = -0.2I[V(s_k, sa_k) = 4] \quad (5)$$

This reward accounts for the cases when the model has made a mistake in understanding the sub-question. Either it has not answered the quantity asked in the sub-question. This gives a signal that the sub-question was either incoherent with the previous questions or was too complex for the model to answer. This kind of mistake is important to address and, hence, is given a higher weight.

**Q** : How many distinct values of  $a, b, c$  are possible?  
**A** : This gives us the side lengths of the triangle as  $a = 10$ ,  $b = 15$ , and  $c = 30$ .  
**Verifier** : Mistakes Understanding Question: The model has made a mistake by not giving the number of distinct values.

**Reward based on place of mistake.** As discussed above, later mistakes are penalised less than

the earlier ones. Hence, if a mistake is made in the first step, it is given a reward of  $-0.2$ . If the model makes a mistake in the first half of the sub-answer, it is given a reward of  $-0.12$ . For a mistake in the last half of the sub-answer, it is given a reward of  $-0.08$ . If the mistake is made in the last step, it is given a reward of  $-0.05$ .

**No-mistake reward** is the case when the model has not made any mistake in answering the sub-question and is given a positive reward of  $+1$ .

## 4 Experiments

For all the experiments,  $LM^2$  uses the OpenAI text-davinci-003 model (hereafter mentioned as GPT-3.5) as the solver and LLaMA-2 13B (Touvron et al., 2023) as the base models for the decomposer and the verifier. Details including the hyperparameters, compute resources, evaluation strategy, etc. are provided in Appendix A.

**Training Data Curation.** For the first stage of fine-tuning of the decomposer LM, we curated a dataset of 15,396 question, concept, sub-question, sub-answer tuples. The questions were taken from the train split of the MATH dataset (Hendrycks et al., 2021). The questions were taken from the MATH dataset. For verifier LM finetuning, a dataset of 3,674 question-answer-classification tuples was generated. Details of the prompts used for each of these steps are provided in Appendix B.

**Baseline Details.** We compare  $LM^2$  with five existing methods: Chain-of-thought prompting (CoT) (Wei et al., 2022), Least-to-most prompting (L2M) (Zhou et al., 2023), Progressive Hint Prompting (PHP) (Zheng et al., 2023), Demonstrate-Search-Predict (DSP) (Khattab et al., 2022), and DaSLaM (Juneja et al., 2023). The original setting of PHP requires an 8-shot prompting; however, since all other methods including  $LM^2$  predict in the zero-shot setting, we use PHP in 1-shot for a fairer comparison.

**Ablation Study.** We perform five types of ablation studies aimed at comprehensively understanding the significance of each component within the  $LM^2$  pipeline: I)  $LM^2 \setminus V$  that ablates the verifier model, II)  $LM^2 \setminus C$ , where we remove the concept generation step, III)  $LM^2 \setminus RL$  removes the PPO fine-tuning, and finally, III)  $LM^2$ -Type and IV)  $LM^2$ -Position are two versions of the framework with rewards that deal with the type of mistakes vs. position of mistakes in the generated response.

Dataset	Method					
	CoT	L2M	PHP	DSP	DaSLaM	LM <sup>2</sup>
PnC	16.4	16.0	10.2	16.2	21.4	<b>30.0</b>
NT	14.4	11.0	9.8	20.3	26.1	<b>41.0</b>
ALG	27.6	22.4	24.0	15.3	33.4	<b>34.0</b>
I-ALG	16.4	16.8	10.0	17.0	24.8	<b>27.8</b>
Calc.	14.0	14.58	14.28	18.8	18.2	<b>34.0</b>
P-ALG	32.3	28.0	26.5	28.0	44.0	<b>47.0</b>
Geom.	14.2	12.5	14.0	5.2	21.4	<b>32.0</b>
MedQA	50.3	49.8	47.5	52.3	50.1	<b>57.1</b>

Table 1: Performance comparison of LM<sup>2</sup> with the baselines on MATH and MedQA datasets using GPT-3.5 as the solver LM.

## 5 Results

We summarize the performance of LM<sup>2</sup> along with the baseline methods on the MATH and MedQA datasets in Table 1 and on the JEEBench dataset in Table 2. Across all the datasets, LM<sup>2</sup> improves upon existing methods (using GPT-3.5 solver) by a huge margin. It demonstrates an average 8% improvement on the MATH dataset and an average 2.5% improvement on the JEEBench dataset as compared to the best-performing baseline DaSLaM.

**Can it improve on out-of-domain tasks?** In both DaSLaM and LM<sup>2</sup>, the solver model is kept frozen with the hope of retaining generalizability. However, the decomposer model in both methods (and the verifier in LM<sup>2</sup>) are finetuned using mathematical reasoning problems. This raises the question of the generalizability of these finetuned components over problems other than mathematical reasoning. One of the most significant challenges with DaSLaM is that it is not able to perform well on out-of-domain tasks like JEEBench Chemistry. We find that our method can surpass this limitation as can be seen in Tables 1 (MedQA) and 2 (JEEBench Chemistry). While DaSLaM degrades the performance over CoT on MedQA, LM<sup>2</sup> achieves an absolute accuracy gain of 6.8 percentage points.

**How important is the verifier?** Next, we seek to investigate the relative importance of each component in our pipeline. We observe that the accuracy decreases substantially upon removing the verifier model (LM<sup>2</sup>\V in the middle third of Table 2). We can see that there is a drop of 13.0% in Chemistry versus 10.08% in Physics and 3.4% in Math subsets. The relative drop in accuracy with the ablation of the verifier is sharper with multi-answer, numeric, and integer answer questions. This makes sense given the computational reasoning requirement is higher in these problems and the verifier plays a crucial role in guiding the decomposer and the solver along the correct reasoning path.

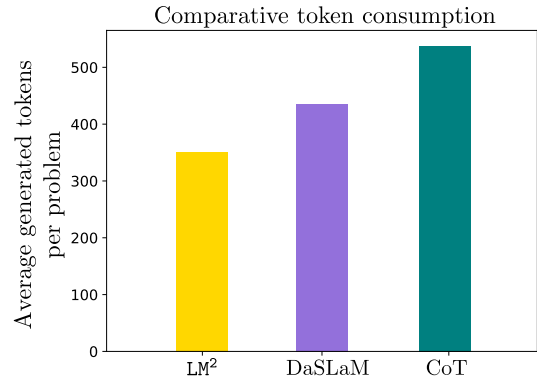


Figure 2: **Comparison of token generation cost.** We depict the average number of tokens generated by the solver model using different methods to solve the given question averaged over 50 questions from the JEEBench dataset.

**How important are the concepts?** As can be seen from Table 2, removing concepts decreases the accuracy of Physics subset by 11.6%, Maths subset by 6.03%, and Chemistry subset by 17.5%. This shows that concepts also play a very important role in improving the performance on out-of-domain datasets like Physics and Chemistry. Typically, LM<sup>2</sup>\C fares worse than the rest of the ablation variants, demonstrating that the concepts are the most important component in LM<sup>2</sup>.

**GPT-4 as concept generator.** We also check how our decomposer compares to GPT-4 while generating concepts. To compare this, we prompt GPT-4 to generate concepts given the question. We observe that there is an average decrease of 9.13% when generating concepts using GPT-4 when compared to the Decomposer model, indicating the higher quality of concepts generated as a result of feedback-based fine-tuning.

**What is the effect of feedback-based finetuning?** The effect of feedback-based fine-tuning is evident when comparing the performance of the decomposer without the second stage of fine-tuning alongside the verifier to that of LM<sup>2</sup>. On average, we observe a notable decrease of 9.6% in performance when the second stage of fine-tuning is omitted. This finding highlights the significance of fine-tuning as a crucial step in optimizing model performance. However, the importance of concepts and the verifier appears to outweigh that of fine-tuning. This suggests that while fine-tuning contributes to improved model performance, the incorporation of concepts and a verifier into the model architecture yields more substantial enhancements.

**How does the structured answering template contribute?** Recall that in LM<sup>2</sup>, we introduce a

Method	Dataset											
	Phy. MCQ	Math. MCQ	Phy. Multi.	Math. Multi.	Phy. Num.	Math. Num.	Phy. Int.	Math. Int.	Chem. Int.	Chem. Num.	Chem. Multi.	Chem. MCQ
CoT	33.33	21.9	6.25	12.0	3.03	1.69	12.5	10.8	17.3	11.6	11.6	40.0
PHP	22.22	17.07	6.25	7.59	3.03	1.69	0*	4.0	11.7	9.7	12.2	37.5
L2M	22.22	21.9	6.25	12.5	3.03	3.38	10.0	10.8	13.0	9.7	10.0	20.0
DaSLaM	<u>55.5</u>	29.5	18.7	16.0	6.06	10.1	15.7	11.7	14.2	9.2	11.6	14.6
GPT4	<b>55.5</b>	<b>34.1</b>	<b>27.5</b>	<b>21.5</b>	<u>15.1</u>	<u>11.8</u>	<b>22.7</b>	<b>24.3</b>	<u>17.9</u>	<b>25.5</b>	<b>48.3</b>	<b>60.0</b>
LM <sup>2</sup>	51.85	30.18	26.8	16.4	<b>15.15</b>	<b>13.1</b>	16.2	13.5	<b>26.0</b>	23.2	26.6	53.3
LM <sup>2</sup> \V	37.03	24.52	14.6	11.7	12.2	11.4	11.4	11.7	17.3	16.2	13.3	30.0
LM <sup>2</sup> \C	29.62	20.75	14.6	9.4	9.09	10.8	9.0	8.1	17.3	11.6	13.3	16.6
GPT4-C	29.62	28.3	14.6	11.5	15.15	11.4	9.0	11.4	21.7	23.2	33.33	30.0
LM <sup>2</sup> \RL	33.33	21.9	18.7	12.7	12.2	10.1	10.0	8.1	17.3	12.4	13.3	27.3
LM <sup>2</sup> -Type	46.1	28.0	20.3	14.0	13.4	11.4	15.0	13.5	24.0	23.2	23.6	45.4
LM <sup>2</sup> -Position	38.4	24.52	16.0	12.9	12.2	11.4	15.0	10.8	24.0	20.6	20.3	33.0
GPT35-SP	33.3	29.2	7.5	12.6	9.0	8.4	12.5	8.0	17.6	9.2	12.2	41.6
GPT4-SP	61.1	36.5	30.0	26.5	30.0	14.2	43.75	32.0	17.6	36.5	49.1	66.6

Table 2: **Performance of LM<sup>2</sup> on JEEBench Dataset along with baselines and ablation variants.** (Top third) we highlight best and second best methods in **boldface** and underline. LM<sup>2</sup> generally outperforms all existing prompting techniques with GPT-3.5 on different topics and different types of questions (other than Physics MCQ questions). In 3/12 cases, LM<sup>2</sup> outperforms GPT-4. (Middle third) We observe a large drop in performance with each ablation variant. (Bottom third) Performance of the structured answer generation employed in LM<sup>2</sup>, without decomposer and verifier, using GPT-3.5 and GPT-4 as solvers.

novel, structured answering template for controllable coordination between the three models. It is imperative to investigate the role of such a template alone behind the performance boost. We make use of the template with two different solver models, GPT-3.5 and GPT-4. As we can see in the bottom third of Table 2 (coined as modelname-SP), both models improve upon their base performance with our structured template. However, the stronger GPT-4 model is able to utilize the template much more efficiently, with an average gain of 7.8% across the JEEBench problems. Typically, improvement on Physics problems is higher than the Math problems, indicating that language models are not very good at retrieving physics concepts and solving the problem when using chain-of-thought prompting. It should be noted that while the structured answering template alone is a powerful boost, it is much weaker alone without the complete coordination in LM<sup>2</sup>.

**Does guided reasoning help limit token usage?** An important challenge with iteratively interacting with an LLM is the increased token usage that will translate to expenses in either computational or monetary terms. In Figure 2, we plot the average token usage (per problem) incurred by the solver model (GPT-3.5) while using LM<sup>2</sup> and DaSLaM against that of base chain-of-thought generation. Note that we only show the token usage corresponding to the modified responses while using LM<sup>2</sup> and DaSLaM. Both these methods originally use base CoT to generate the initial response

and therefore, their total token usage will always be higher than that of CoT. However, the added structure and guided reasoning significantly reduce the token usage in the modified response. LM<sup>2</sup> prevails in this aspect too. A major reason behind this is the step-by-step synergy between the decomposer, the solver, and the verifier in LM<sup>2</sup>. Since the decomposer generates the subquestion depending upon the response from the solver to the previous subquestion, the chances of redundant generation decrease, as opposed to DaSLaM where the subquestions are generated all at once.

**Example analysis.** To further understand the nuances of LM<sup>2</sup>, we perform an analysis of the generated output on an example from the MATH dataset (see Figure 3). We compare between LM<sup>2</sup>, DaSLaM and GPT-4 with CoT. As we can see, GPT-4 makes an incorrect interpretation of the question itself. It assumes that the total journey after delay takes 10 hours, leading to an incorrect choice of option. The subquestions produced by DaSLaM do not adhere to the order of reasoning required to solve the problem and generate redundant questions. It starts with asking *What is the total distance to be covered?* However, in the second question, it asks for the speed of the train which is already given in the question itself. The 3rd subquestion generated by DaSLaM is actually the original question, and the solver makes a numerical mistake by simplifying the fraction  $\frac{3d}{75}$  to  $\frac{d}{300}$  instead of  $\frac{d}{100}$ . Without a verifier, this erroneous response is integrated into the reasoning context of the solver. In the next ques-

A train running at a speed of 100 miles/hour, takes 10 hours to reach its destination. After covering quarter of the distance, it starts raining and the train has to be slowed to speed of 75 miles/hour. What is the total journey duration? )10 B)11.5 C)12.5 D)13.5 E)1

GPT 4	DaSLaM	LM <sup>2</sup>
<p>Let's denote the total distance of the journey as <math>D</math> miles. Initially, the train runs at a speed of 100 miles/hour. So, in the first part of the journey, it covers <math>\frac{1}{4}</math> of the total distance at this speed. The time taken to cover this distance can be calculated using the formula:</p> $\text{Time}_1 = \frac{\frac{1}{4}D}{100} = \frac{D}{400}$ <p>Now, after covering <math>\frac{1}{4}</math> of the distance, the remaining distance to be covered is <math>\frac{3}{4}D</math>. When it starts raining, the train slows down to a speed of 75 miles/hour. So, the time taken to cover the remaining distance at this speed is:</p> $\text{Time}_2 = \frac{\frac{3}{4}D}{75} = \frac{D}{100}$ <p>The total journey duration is the sum of the time taken for the first part of the journey and the time taken for the second part:</p> $\text{Total journey duration} = \text{Time}_1 + \text{Time}_2 = \frac{D}{400} + \frac{D}{100}$ <p>Now, let's factor out <math>D</math>:</p> $\text{Total journey duration} = D \left( \frac{1}{400} + \frac{1}{100} \right)$ $\text{Total journey duration} = D \left( \frac{1}{400} + \frac{4}{400} \right)$ $\text{Total journey duration} = D \left( \frac{1+4}{400} \right)$ $\text{Total journey duration} = D \left( \frac{5}{400} \right)$ $\text{Total journey duration} = \frac{5D}{400}$ <p>Given that the total journey duration is 10 hours, we can set up the equation:</p> $10 = \frac{5D}{400}$ <p>Now, solve for <math>D</math>:</p> $D = 10 \times \frac{400}{5}$ $D = 80$ <p>So, the total distance of the journey is 80 miles. Now, we can find the total journey duration:</p> $\text{Total journey duration} = \frac{5 \times 80}{400}$ $\text{Total journey duration} = \frac{400}{400}$ $\text{Total journey duration} = 1 \text{ hour}$ <p>Therefore, the total journey duration is 1 hour + 10 hours = 11 hours. So, the correct answer is option B) 11.5.</p>	<p><b>Sub problem 1:</b> What is the total distance to be covered?</p> <p><b>Sub Answer 1:</b> Let the total distance to be covered be <math>d</math>. Time taken to cover the first quarter of the distance = <math>\frac{1}{4} \frac{d}{100} = \frac{d}{400}</math> hours. Time taken to cover the remaining distance = <math>\frac{3}{4} \frac{d}{75} = \frac{300}{400}</math> hours. Total time taken = <math>\frac{d}{400} + \frac{300}{400} = \frac{10}{4}</math> hours = 2.5 hours. Since the train takes 10 hours to reach its destination, Total distance to be covered = <math>10 \times 100 = 1000</math> miles.</p> <p><b>Sub problem 2:</b> What is the speed of the train?</p> <p><b>Sub Answer 2:</b> After covering quarter of the distance, the train has to be slowed to speed of 75 miles/hour. Let the remaining distance be <math>x</math>. Therefore, <math>x = d - \frac{d}{4} = 1000 - \frac{1000}{4} = 750</math> miles. The time taken to cover the remaining distance <math>x = 750</math> miles at a speed of 75 miles/hour = <math>\frac{750}{75}</math> hours = 10 hours. Therefore, the speed of the train = <math>\frac{x}{\text{time taken}} = \frac{750}{10} = 75</math> miles/hour.</p> <p><b>Sub problem 3:</b> What is the time taken to cover the distance?</p> <p><b>Sub Answer 3:</b> Time taken to cover the first quarter of the distance = <math>\frac{d}{400}</math> hours = <math>\frac{d}{400}</math> hours. Time taken to cover the remaining distance = <math>\frac{3d}{75}</math> hours = <math>\frac{d}{25}</math> hours. Total time taken = <math>\frac{d}{400} + \frac{d}{25} = \frac{d+16d}{400} = \frac{17d}{400}</math> hours. Putting <math>d = 1000</math>, Final answer: The time taken to cover the distance is <math>\frac{17 \times 1000}{400}</math> hours.</p> <p><b>Sub problem 4:</b> What is the total duration of the journey?</p> <p><b>Sub Answer 4:</b> Time taken to cover the first quarter of the distance = <math>\frac{d}{400}</math> hours = <math>\frac{d}{400}</math> hours. Time taken to cover the remaining three quarters of the distance = <math>\frac{3d}{75}</math> hours = <math>\frac{4d}{25}</math> hours. Total time taken = <math>\frac{d}{400} + \frac{4d}{25} = \frac{d+64d}{400} = \frac{65d}{400}</math> hours. Total duration of the journey = <math>\frac{65d}{400}</math> hours = 9 hours and <math>\frac{d}{700}</math> hours = 9 hours and 14.29 minutes.</p> <p><b>Final Answer:</b></p> <p>Step 1: The train covers the first quarter of the journey at a speed of 100 miles/hour.</p> <p>Step 2: The time taken to cover the first quarter of the journey is given by: Time = Distance/Speed Time = <math>\frac{1}{4} \times \frac{1000}{100}</math> Time = <math>\frac{1}{4}</math> hour</p> <p>Step 3: The train covers the remaining three quarters of the journey at a speed of 75 miles/hour.</p> <p>Step 4: The time taken to cover the remaining three quarters of the journey is given by: Time = Distance/Speed Time = <math>\frac{3}{4} \times \frac{1000}{75}</math> Time = <math>\frac{3}{4}</math> hour</p> <p>Step 5: The total duration of the journey is the sum of the time taken to cover the first quarter and the remaining three quarters of the journey.</p> <p>Total Time = <math>\frac{1}{4} + \frac{3}{4}</math> Total Time = <math>\frac{4}{4}</math> Total Time = 1 hour</p> <p>Therefore, the total duration of the journey is 1 hour.</p>	<p><b>Required Concepts:</b></p> <p>Kinematics Average Speed Distance formula Time formula</p> <p><b>Relevant Equations:</b></p> <p>Average Speed = Total Distance / Total Time Distance = Speed <math>\times</math> Time Time = Distance / Speed</p> <p><b>Subquestions:</b></p> <ul style="list-style-type: none"> <li>What is the total distance traveled by the train?</li> <li>How much time does it take for the train to cover the first quarter of the distance?</li> <li>What is the remaining distance to be covered after the train slows down?</li> <li>How much time does it take for the train to cover the remaining distance at the reduced speed?</li> <li>What is the total journey duration?</li> </ul> <p><b>Subanswers:</b></p> <ul style="list-style-type: none"> <li>Total distance = Initial speed <math>\times</math> Total time = 100 miles/hour <math>\times</math> 10 hours = 1000 miles</li> <li>Time taken to cover the first quarter of the distance = Distance / Speed = 1000 miles / 4 / 100 miles/hour = 2.5 hours</li> <li>Remaining distance after covering the first quarter = Total distance - Distance covered = 1000 miles - 250 miles = 750 miles</li> <li>Time taken to cover the remaining distance at reduced speed = Remaining distance / Reduced speed = 750 miles / 75 miles/hour = 10 hours</li> <li>Total journey duration = Time taken to cover first quarter + Time taken to cover remaining distance = 2.5 hours + 10 hours = 12.5 hours</li> </ul> <p>Final Answer: The total journey duration is 12.5 hours.</p>

Figure 3: Comparison of GPT-4, DaSLaM and LM<sup>2</sup> on an example from the MATH dataset.

539 tions, the same problem is asked to be solved and  
540 the solver continues to make incorrect responses.  
541 With LM<sup>2</sup>, we observe a much more well-defined,  
542 crisp line of questioning by the decomposer model;  
543 the solver is able to reach the correct answer op-  
544 tion without regenerating the same information or  
545 drawing incorrect subanswers.

## 6 Conclusion

547 In this paper, we present LM<sup>2</sup>, a cooperative cohort  
548 of generative language models working together to  
549 solve complex reasoning problems. LM<sup>2</sup> utilizes a  
550 frozen solver model that is guided to solve reason-

ing problems by incrementally answering questions  
framed by a decomposer model and checked by the  
verifier model that is trained to coordinate with  
each other. We find that LM<sup>2</sup> proves its supremacy  
over existing methods over a variety of reasoning  
tasks, both in-domain and out-domain. We find that  
despite being trained using mathematical reason-  
ing examples, our proposed structured response  
scheme along with the fine-grained verification  
strategy plays a crucial role in generalizing LM<sup>2</sup>  
to heavily out-of-distribution tasks like medical  
question answering and chemistry.

551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562



## 563 **Limitations**

564 Despite promising results, LM<sup>2</sup> bears some inherent  
565 limitations. Compared to purely prompting-based  
566 methods, it requires a certain computational over-  
567 head for the two-staged training. With proprietary  
568 LLM-based solvers, LM<sup>2</sup> incurs extra token usage  
569 over single-pass solutions like chain-of-thought.  
570 Implicit limitations of the solver model, like lack  
571 of length generalization, arbitrary digit manipula-  
572 tion, etc. are expected to be inherited in LM<sup>2</sup> as  
573 well. A possible future work can be towards in-  
574 corporating deterministic solvers and tools into the  
575 multiplex.

## 576 **Ethical considerations**

577 The research results are communicated honestly  
578 and credibly and transparency has been maintained  
579 throughout the research process. Innovations con-  
580 cerning automated problem-solving bear important  
581 ramifications in terms of human labor-value. This  
582 research is no exception in terms of future appli-  
583 cation. This research relies on the usage of pro-  
584 prietary models for building datasets and running  
585 experiments. The inequality in accessibility of such  
586 resources among research communities might af-  
587 fect future usage of this work.

## 588 **References**

589 Daman Arora, Himanshu Gaurav Singh, and Mausam.  
590 2023. [Have llms advanced enough? a challenging  
591 problem solving benchmark for large language mod-  
592 els.](#) *Preprint*, arXiv:2305.15074.

593 Wenhui Chen, Xueguang Ma, Xinyi Wang, and  
594 William W Cohen. 2022. Program of thoughts  
595 prompting: Disentangling computation from reason-  
596 ing for numerical reasoning tasks. *arXiv preprint  
597 arXiv:2211.12588*.

598 Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian,  
599 Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias  
600 Plappert, Jerry Tworek, Jacob Hilton, Reiichiro  
601 Nakano, Christopher Hesse, and John Schulman.  
602 2021. [Training verifiers to solve math word prob-  
603 lems.](#) *Preprint*, arXiv:2110.14168.

604 Subhabrata Dutta, Ishan Pandey, Joykirat Singh, Sunny  
605 Manchanda, Soumen Chakrabarti, and Tanmoy  
606 Chakraborty. 2024. Frugal llms trained to invoke sym-  
607 bolic solvers achieve parameter-efficient arithmetic  
608 reasoning. In *Proceedings of the AAAI Conference  
609 on Artificial Intelligence*, volume 38, pages 17951–  
610 17959.

611 Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul  
612 Arora, Steven Basart, Eric Tang, Dawn Song, and

Jacob Steinhardt. 2021. Measuring mathematical  
613 problem solving with the math dataset. *NeurIPS*. 614

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-  
615 Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu  
616 Chen. 2022. [LoRA: Low-rank adaptation of large  
617 language models.](#) In *International Conference on  
618 Learning Representations*. 619

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng,  
620 Hanyi Fang, and Peter Szolovits. 2020. [What dis-  
621 ease does this patient have? a large-scale open do-  
622 main question answering dataset from medical exams.](#)  
623 *Preprint*, arXiv:2009.13081. 624

Gurusha Juneja, Subhabrata Dutta, Soumen Chakrabarti,  
625 Sunny Manchanda, and Tanmoy Chakraborty. 2023.  
626 [Small language models fine-tuned to coordinate  
627 larger language models improve complex reasoning.](#)  
628 In *Proceedings of the 2023 Conference on Empiri-  
629 cal Methods in Natural Language Processing*, pages  
630 3675–3691, Singapore. Association for Computa-  
631 tional Linguistics. 632

Omar Khattab, Keshav Santhanam, Xiang Lisa  
633 Li, David Hall, Percy Liang, Christopher Potts,  
634 and Matei Zaharia. 2022. [Demonstrate-search-  
635 predict: Composing retrieval and language mod-  
636 els for knowledge-intensive nlp.](#) *arXiv preprint  
637 arXiv:2212.14024*. 638

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao  
639 Fu, Kyle Richardson, Peter Clark, and Ashish Sab-  
640 harwal. 2023. [Decomposed prompting: A modular  
641 approach for solving complex tasks.](#) In *The Eleventh  
642 International Conference on Learning Representa-  
643 tions*. 644

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen,  
645 Jian-Guang Lou, and Weizhu Chen. 2023. [Making  
646 large language models better reasoners with step-  
647 aware verifier.](#) *Preprint*, arXiv:2206.02336. 648

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari,  
649 Henryk Michalewski, Jacob Austin, David Bieber,  
650 David Dohan, Aitor Lewkowycz, Maarten Bosma,  
651 David Luan, et al. 2021. Show your work: Scratch-  
652 pads for intermediate computation with language  
653 models. *arXiv preprint arXiv:2112.00114*. 654

OpenAI. 2023. [Gpt-4 technical report.](#) *Preprint*,  
655 arXiv:2303.08774. 656

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec  
657 Radford, and Oleg Klimov. 2017. [Proximal policy  
658 optimization algorithms.](#) *CoRR*, abs/1707.06347. 659

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu,  
660 Junxiao Song, Mingchuan Zhang, YK Li, Y Wu, and  
661 Daya Guo. 2024. [Deepseekmath: Pushing the limits  
662 of mathematical reasoning in open language models.](#)  
663 *arXiv preprint arXiv:2402.03300*. 664

Hao Sun, Hengyi Cai, Bo Wang, Yingyan Hou, Xi-  
665 aochi Wei, Shuaiqiang Wang, Yan Zhang, and Dawei  
666 Yin. 2023. [Towards verifiable text generation with](#)  
667

668 evolving memory and self-reflection. *arXiv preprint*  
669 *arXiv:2312.09075*.

670 Denis Tarasov and Kumar Shridhar. 2024. [Distilling](#)  
671 [llms’ decomposition abilities into compact language](#)  
672 [models](#). *Preprint*, arXiv:2402.01812.

673 Shubham Toshniwal, Ivan Moshkov, Sean Narenthi-  
674 ran, Daria Gitman, Fei Jia, and Igor Gitman. 2024.  
675 Openmathinstruct-1: A 1.8 million math instruction  
676 tuning dataset. *arXiv preprint arXiv:2402.10176*.

677 Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier  
678 Martinet, Marie-Anne Lachaux, Timothée Lacroix,  
679 Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal  
680 Azhar, Aurelien Rodriguez, Armand Joulin, Edouard  
681 Grave, and Guillaume Lample. 2023. [Llama: Open](#)  
682 [and efficient foundation language models](#). *Preprint*,  
683 arXiv:2302.13971.

684 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten  
685 Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le,  
686 and Denny Zhou. 2022. [Chain of thought prompt-](#)  
687 [ing elicits reasoning in large language models](#). In  
688 *Advances in Neural Information Processing Systems*.

689 Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He,  
690 Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao.  
691 2023. [Large language models are better reasoners](#)  
692 [with self-verification](#). *Preprint*, arXiv:2212.09561.

693 Zhuofeng Wu, He Bai, Aonan Zhang, Jiatao Gu,  
694 VG Vinod Vydiswaran, Navdeep Jaitly, and Yizhe  
695 Zhang. 2024. [Divide-or-conquer? which part should](#)  
696 [you distill your llm?](#) *Preprint*, arXiv:2402.15000.

697 Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran,  
698 Tom Griffiths, Yuan Cao, and Karthik Narasimhan.  
699 2024. Tree of thoughts: Deliberate problem solving  
700 with large language models. *Advances in Neural*  
701 *Information Processing Systems*, 36.

702 Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo  
703 Li, and Yu Li. 2023. [Progressive-hint prompting im-](#)  
704 [proves reasoning in large language models](#). *Preprint*,  
705 arXiv:2304.09797.

706 Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei,  
707 Nathan Scales, Xuezhi Wang, Dale Schuurmans,  
708 Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H.  
709 Chi. 2023. [Least-to-most prompting enables com-](#)  
710 [plex reasoning in large language models](#). In *The*  
711 *Eleventh International Conference on Learning Rep-*  
712 *resentations*.

## 713 A Training Details and Evaluation

714 We finetune LLaMA2-13B for both the decomposer  
715 and verifier. We train for 8 epochs with a batch size  
716 of 128, learning rate 2e-5, warmup steps of 100, a  
717 Lora r value of 4, LoRA Alpha of 16 and dropout of  
718 0.05. The models were trained in 8-bit quantization  
719 on an 80G A100 GPU.

720 For the second stage of fine-tuning, we finetuned  
721 the last 3 layers of LoRA adapters, using a batch  
722 size of 16, gradient accumulation steps=4, init kl  
723 coef=0.01, target=4. For inference, we used a tem-  
724 perature of 0 in all experiments for consistency of  
725 results with a max output length of 2000.

726 We evaluate our method on hard reasoning  
727 datasets that require multi-step reasoning. These  
728 datasets include MATH (Hendrycks et al., 2021)  
729 (test split), JEEBench (Arora et al., 2023), and  
730 MedQA (Jin et al., 2020) (English questions). The  
731 MATH dataset contains math questions from chal-  
732 lenging math competitions, since it was also used  
733 for training, this shows our performance on in-  
734 domain questions. Next, we evaluate on the out-of-  
735 distribution datasets like JEEBench which contains  
736 PCM questions extracted from the JEE Advanced  
737 exam and MedQA which contains open-domain  
738 questions from professional medical board exams.  
739 We only evaluate questions in the English language.

## 740 B Training Data Creation

741 The data was generated using GPT-4. A tempera-  
742 ture of 0.7 is used to ensure diversity in the gen-  
743 erated data. We only stored the sub-question, sub-  
744 answer dataset if the number of sub-questions gen-  
745 erated was more than three, this was done to ensure  
746 high data quality so that the model is able to decom-  
747 pose longer and more difficult questions effectively.  
748 First, we generate all the concepts, then the sub-  
749 questions given the question and the gold chain  
750 of thought. Finally, we generate the sub-answer  
751 given the question, a gold chain of thought and  
752 the sub-question to be answered. for the verifier,  
753 we first ask the LLM to answer the given question  
754 using standard COT prompting. Then based on  
755 the correctness of the answer, we take the solution  
756 chain of thought produced by the LLM and the gold  
757 answer and ask the LLM to classify the produced  
758 solution based on the mistake made. If the answer  
759 is correct, we store it separately and include it to  
760 make up to 10% of the dataset with the label as ‘No  
761 Mistake’. Prompts for the data curation are given  
762 below.

### 763 B.1 Verifier Data Creation

#### 764 B.1.1 Prompt

765 You are a teacher, and you are grading  
766 a student’s answer to a question. The  
767 student’s answer is as follows: {COT\_LLM}  
768 The correct answer is as follows:

769 {COT\_gold} Please provide feedback to  
770 the students on the mistakes they have  
771 made. You need to fill out a rubric and  
772 classify the mistakes into the following  
773 categories:

774 1. Conceptual Mistakes: The student has  
775 misunderstood the concept or has applied  
776 the wrong concept.

777 2. Computational Mistakes: The student  
778 has made a mistake in the calculations.

779 3. Procedural Mistakes: The student has  
780 made a mistake in the procedure of solving  
781 the problem.

782 4. Mistake in understanding the question:  
783 The student has made a mistake in  
784 understanding the question.

785 5. Mistake in the first step: The student  
786 has made a mistake in the first step of  
787 the solution.

788 6. Mistake in the first half: The student  
789 has made a mistake in the first half of  
790 the solution.

791 7. Mistake in the second half: The student  
792 has made a mistake in the second half of  
793 the solution.

794 8. Mistake in the last step: The student  
795 has made a mistake in the last step of  
796 the solution.

797 9. No mistake: The student has not made  
798 any mistake.

799 Please first provide feedback then  
800 fill the rubric and then finally tell  
801 your feedback to the student in between  
802 <feedback> and </feedback> tags as shown  
803 below:

804 For example, if you want to tell the  
805 student that they have made a mistake in  
806 the first step and a conceptual mistake,  
807 then you need to write the following:  
808 <feedback> 1,4 </feedback> Do not write  
809 anything else in between <feedback> and  
810 </feedback> tags except the numbers.

811 Now, please provide feedback to the  
812 student on the mistakes they have made.

## 813 B.2 Decomposer Data Creation

### 814 B.2.1 Concepts data creation

815 I have a question's solution, tell me  
816 all the specific concepts, theorems and  
817 formulas (separated by a comma,) used in  
818 it. An example is given below.

Question: How many primes are in the row  
of Pascal's Triangle that starts with a 1  
followed by a 6?

Answer: If the row contains a 1, then  
a 6, then the binomial coefficients must  
be  $\binom{6}{0}$  and  $\binom{6}{1}$ . All we need to check now  
are  $\binom{6}{2}$  and  $\binom{6}{3}$ , since  $\binom{6}{0} = \binom{6}{6}$ ,  $\binom{6}{1} = \binom{6}{5}$   
, and  $\binom{6}{2} = \binom{6}{4}$ .  $\binom{6}{2} = \frac{6!}{4! \times 2!} = 15$ , and  
 $\binom{6}{3} = \frac{6!}{3! \times 3!} = 20$ . None of those is prime,  
so there are  prime numbers in the given  
row.

Concepts: Coefficients in Pascal's  
Triangle, Binomial Coefficients Formula,  
Prime Numbers

Question: question

Answer: answer

Concepts:

### 836 B.2.2 Sub-question data creation

I have a question, it's a solution and a  
sub-question.

Your task is to break the question into  
sub-questions based on the steps in the  
answer.

Keep the following tips in mind:

1. Make sure not to break the  
question into trivial sub-questions, the  
sub-questions should be informative.

2. The sub-questions should not require  
multiple steps to answer, something like  
2-3 steps to solve is ideal.

3. One way to break the question could  
be to identify what all quantities are  
required in the question by observing it's  
answer and then try to frame sub-questions  
based on the unknown entities.

4. Make sure to put each question in the  
question tag like \$ question(What is the  
acceleration of the car as a function of  
time?)\$

One example is given below.

Question: How many primes are in the row  
of Pascal's Triangle that starts with a 1  
followed by a 6?

Answer: If the row contains a 1, then  
a 6, then the binomial coefficients must  
be  $\binom{6}{0}$  and  $\binom{6}{1}$ . All we need to check now  
are  $\binom{6}{2}$  and  $\binom{6}{3}$ , since  $\binom{6}{0} = \binom{6}{6}$ ,  $\binom{6}{1} = \binom{6}{5}$   
, and  $\binom{6}{2} = \binom{6}{4}$ .  $\binom{6}{2} = \frac{6!}{4! \times 2!} = 15$ , and  
 $\binom{6}{3} = \frac{6!}{3! \times 3!} = 20$ . None of those is prime,  
so there are  prime numbers in the given

869 row.  
870 Sub-questions:  
871 \$ question(How can the first two numbers  
872 be represented in form of binomial  
873 coefficients?)\$, \$ question(What are the  
874 values of all the coefficients in the  
875 row?)\$, \$ question(How many of the above  
876 numbers are prime?)\$  
877 Question: question  
878 Answer: answer  
879 Sub-questions:

### 880 **B.3 Sub-answer data generation**

881 I have a question, it's solution and a  
882 sub-question.  
883 I want you to answer the subquestion along  
884 with an explanation.  
885 Make sure to put the sub-answer in  
886 the answer tag like \$sub-answer(The  
887 acceleration of the car at time t =  
888 2 seconds is speed / time = 2m/s/2s =  
889  $1m/s^2$ )\$  
890 Think step by step.  
891 Question: question  
892 Answer: answer  
893 Sub-question: sub-question-array[i]  
894 Sub-Answer: