LM^2: A Simple Society of Language Models Solves Complex Reasoning

Gurusha Juneja*
Microsoft Research, India
gurushajuneja@gmail.com

Subhabrata Dutta
IIT Delhi, India
subha0009@gmail.com

Tanmoy Chakraborty
IIT Delhi, India
tanchak@ee.iitd.ac.in

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^2 to address these challenges. LM^2 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^2 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 (code available at https://github.com/ LCS2-IIITD/Language_Model_Multiplex).

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; Khattab 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, **L**anguage **M**odel **M**ultiplex (LM^2). LM^2 is built upon three separate LMs, each dedicated to

*Equal contribution

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 16473–16484
November 12-16, 2024

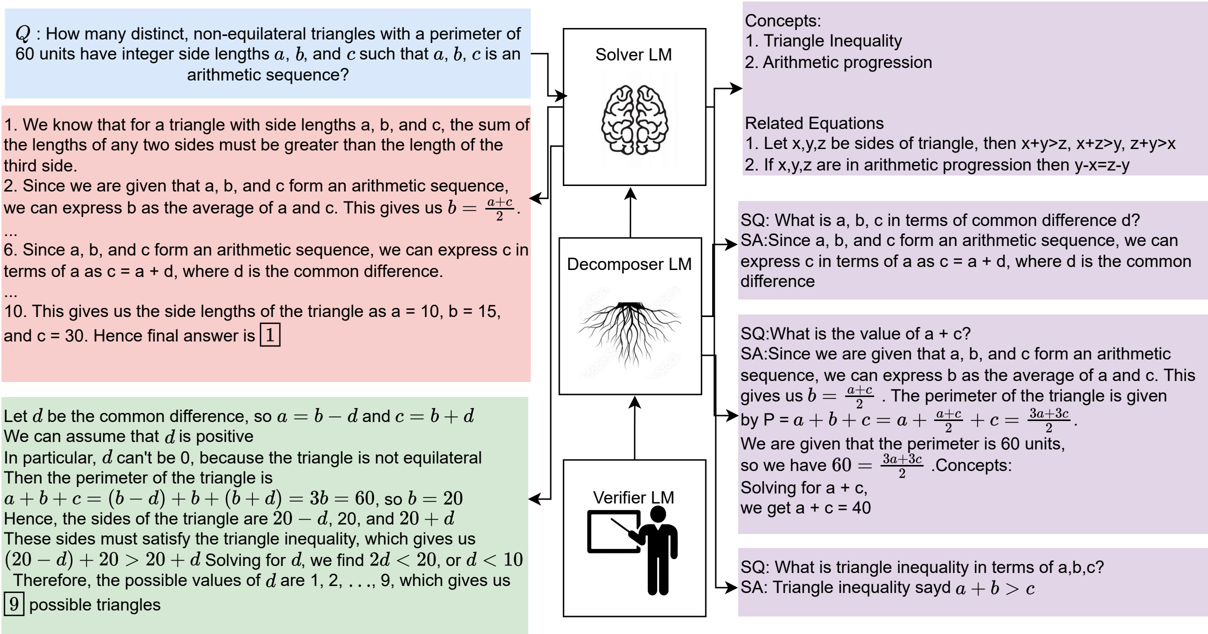

Figure 1: The inference procedure of LM^2 on a question from the MATH dataset. A question (in blue) is provided to the Solver LM that produces an incorrect answer (in red). The question is then provided to the Decomposer LM that generates the concepts and step-by-step subquestions (in lilac). Each subquestion 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 subqustion-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).

three different components of complex multistep reasoning – a **solver** LM is responsible for answering questions; a **verifier** LM provides feedback on the correctness of the output from the solver, and a **decomposer** LM identifies the basic concepts required to solve the problem and generates step-by-step subproblems by decomposing the original question (see Figure 1 for a working example). Unlike prior approaches, the decomposer in LM^2 generates each subproblem depending on the solver's answers to prior subproblems, along with the verifier's feedback on those answers. Furthermore, the decomposer generates the conceptual requirements to solve the problem, which further streamlines the solver LM. Irrespective of the complexity of the underlying reasoning, the world knowledge required to answer any question is typically better preserved in larger, proprietary LMs. Considering this, we use GPT-3.5 (`text-davinci-003`) as the solver without finetuning. For both the decomposer and verifier, we implement parameter-efficient finetuning (Hu et al., 2022) of LLaMA-2 (13 billion parameters) separately. First, these models are finetuned separately towards the tasks of decomposition and verification using datasets annotated by GPT-4. The decomposer is then taught to coordinate with the solver and the verifier models in a policy learning setup. LM^2 achieves promising performance across a diverse set of reasoning tasks. On the MATH dataset of mathematical reasoning, LM^2 outperforms the best decomposer-tuning baseline by a staggering margin 8.1% of absolute accuracy on average. Although LM^2 uses the training split of the MATH dataset for tuning the decomposer 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 observe that (i) the verifier LM and concept generated by the decomposer LM play a crucial role in generalizing out-of-distribution reasoning tasks like MedQA, JEEBench Chemistry, etc.; (ii) finetuning 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.

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`^{2}, 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`^{2} 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`^{2}, we make use of a structured, step-by-step input-output framework (see Figure 1). The role of each of the modules in `LM`^{2} 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 utilize 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 Θ. The model Θ 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 ϕ 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 ϕ. 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:

$$R = \gamma^k \sum_{i=1}^{9} r_i \tag{1}$$

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,

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

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 $+1$.

We perform a hyperparameter search for each class of reward.

4 Experiments

For all the experiments, $\texttt{LM}^2$ 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 finetuning of the decomposer LM, we curated a dataset of 15,396 question, concept, sub-question, subanswer 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 $\texttt{LM}^2$ 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 $\texttt{LM}^2$ 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

Dataset	Method					
	CoT	L2M	PHP	DSP	DaSLaM	LM²
PnC	16.4	16.0	10.2	16.2	21.4	**30.0**
NT	14.4	11.0	9.8	20.3	26.1	**41.0**
ALG	27.6	22.4	24.0	15.3	33.4	**34.0**
I-ALG	16.4	16.8	10.0	17.0	24.8	**27.8**
Calc.	14.0	14.58	14.28	18.8	18.2	**34.0**
P-ALG	32.3	28.0	26.5	28.0	44.0	**47.0**
Geom.	14.2	12.5	14.0	5.2	21.4	**32.0**
MedQA	50.3	49.8	47.5	52.3	50.1	**57.1**

Table 1: Performance comparison of LM² with the baselines on MATH and MedQA datasets using GPT-3.5 as the solver LM.

the significance of each component within the LM² pipeline: I) LM²\V that ablates the verifier model, II) LM²\C, where we remove the concept generation step, III) LM²\RL removes the PPO fine-tuning, and finally, III) LM²-Type and IV) LM²-Position are two versions of the framework with rewards that deal with the type of mistakes vs. position of mistakes in the generated response.

5 Results

We summarize the performance of LM² 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, LM² 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 LM², the solver model is kept frozen with the hope of retaining generalizability. However, the decomposer model in both methods (and the verifier in LM²) 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, LM² 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 (LM²\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.

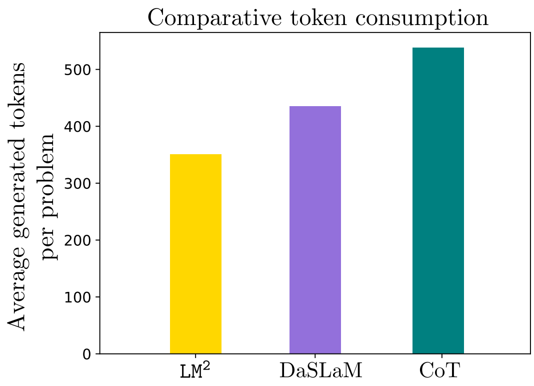

Figure 2: **Comparison of token generation cost.** We depict the average number of tokens generated by the solver model using different methods to solve the given question averaged over 50 questions from the JEEBench dataset.

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, LM²\C fares worse than the rest of the ablation variants, demonstrating that the concepts are the most important component in LM².

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 LM². 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-

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	55.5	29.5	18.7	16.0	6.06	10.1	15.7	11.7	14.2	9.2	11.6	14.6
GPT4	**55.5**	**34.1**	**27.5**	**21.5**	15.1	11.8	**22.7**	**24.3**	17.9	**25.5**	**48.3**	**60.0**
LM^2	51.85	30.18	26.8	16.4	**15.15**	**13.1**	16.2	13.5	**26.0**	23.2	26.6	53.3
LM^2\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^2\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^2\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^2-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^2-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^2 on JEEBench Dataset along with baselines and ablation variants**. (*Top third*) we highlight best and second best methods in **boldface** and underline. LM^2 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^2 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^2, without decomposer and verifier, using GPT-3.5 and GPT-4 as solvers.

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^2, we introduce a 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 noted that while the structured answering template alone is a powerful boost, it is much weaker alone without the complete coordination in LM^2.

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^2 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^2 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^2 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^2. 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.

Does LM^2 help improve the reasoning capability of smaller models? Generalization ability of an algorithm with respect to the solver models used is important. To demonstrate that LM^2 is model agnostic, in Table 3 we experiment LM^2 with Mixtral 8x7B and GPT-3.5 Turbo as solver models while the decomposer and verifier remain the same trained models. We can observe strong improvements achieved over the base models when paired with LM2.

Effect of finetuning the decomposer and veri-

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

Let's denote the total distance of the journey as D miles.
Initially, the train runs at a speed of 100 miles/hour. So, in the first part of the journey, it covers $\frac{1}{4}$ of the total distance at this speed. The time taken to cover this distance can be calculated using the formula:

$$\text{Time}_1 = \frac{\frac{1}{4}D}{100} = \frac{D}{400}$$

Now, after covering $\frac{1}{4}$ of the distance, the remaining distance to be covered is $\frac{3}{4}D$.
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:

$$\text{Time}_2 = \frac{\frac{3}{4}D}{75} = \frac{D}{100}$$

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:

$$\text{Total journey duration} = \text{Time}_1 + \text{Time}_2 = \frac{D}{400} + \frac{D}{100}$$

Now, let's factor out D:

$$\text{Total journey duration} = D\left(\frac{1}{400} + \frac{1}{100}\right)$$

$$\text{Total journey duration} = D\left(\frac{1}{400} + \frac{4}{400}\right)$$

$$\text{Total journey duration} = D\left(\frac{1+4}{400}\right)$$

$$\text{Total journey duration} = D\left(\frac{5}{400}\right)$$

$$\text{Total journey duration} = \frac{5D}{400}$$

Given that the total journey duration is 10 hours, we can set up the equation:

$$10 = \frac{5D}{400}$$

Now, solve for D:

$$D = 10 \times \frac{400}{5}$$

$$D = 80$$

So, the total distance of the journey is 80 miles.
Now, we can find the total journey duration:

$$\text{Total journey duration} = \frac{5 \times 80}{400}$$

$$\text{Total journey duration} = \frac{400}{400}$$

$$\text{Total journey duration} = 1 \text{ hour}$$

Therefore, the total journey duration is 1 hour + 10 hours = 11 hours.
So, the correct answer is option B) 11.5.

DaSLaM

Sub problem 1: What is the total distance to be covered?

Sub Answer 1: Let the total distance to be covered be d.
Time taken to cover the first quarter of the distance = $\frac{1}{4}\frac{d}{100} = \frac{10}{400}$ hours.
Time taken to cover the remaining distance = $\frac{3}{4}\frac{d}{75} = \frac{300}{400}$ hours.
Total time taken = $\frac{10}{400} + \frac{300}{400} = \frac{10}{4}$ hours = 2.5 hours.
Since the train takes 10 hours to reach its destination,
Total distance to be covered = $10 \times 100 = 1000$ miles.

Sub problem 2: What is the speed of the train?

Sub Answer 2: After covering quarter of the distance, the train has to be slowed to speed of 75 miles/hour.
Let the remaining distance be x.
Therefore, $x = d - \frac{d}{4} = 1000 - \frac{1000}{4} = 750$ miles.
The time taken to cover the remaining distance $x = 750$ miles at a speed of 75 miles/hour = $\frac{750}{75}$ hours = 10 hours.
Therefore, the speed of the train = $\frac{x}{\text{time taken}} = \frac{750}{10} = 75$ miles/hour.

Sub problem 3: What is the time taken to cover the distance?

Sub Answer 3: Time taken to cover the first quarter of the distance = $\frac{\frac{d}{4}}{100}$ hours = $\frac{d}{400}$ hours.
Time taken to cover the remaining distance = $\frac{\frac{3d}{4}}{75}$ hours = $\frac{d}{300}$ hours.
Total time taken = $\frac{d}{400} + \frac{d}{300} = \frac{5d+4d}{1200}$ hours = $\frac{9d}{1200}$ hours.
Putting $d = 1000$,
Final answer: The time taken to cover the distance is $\frac{9\times1000}{1200}$ hours.

Sub problem 4: What is the total duration of the journey?

Sub Answer 4: Time taken to cover the first quarter of the distance = $\frac{\frac{d}{4}}{100}$ hours = $\frac{d}{400}$ hours.
Time taken to cover the remaining three quarters of the distance = $\frac{\frac{3d}{4}}{75}$ hours = $\frac{4d}{300}$ hours.
Total time taken = $\frac{d}{400} + \frac{4d}{300} = \frac{5d+4d}{(400+300)} = \frac{9d}{700}$ hours.
Total duration of the journey = $\frac{9d}{700}$ hours = 9 hours and $\frac{d}{700}$ hours = 9 hours and $\frac{100}{700}$ hours = 9 hours and 14.29 minutes.

Final Answer:

Step 1: The train covers the first quarter of the journey at a speed of 100 miles/hour.

Step 2: The time taken to cover the first quarter of the journey is given by: Time = Distance/Speed
Time = $\frac{1}{4} \times \frac{100}{100}$ Time = $\frac{1}{4}$ hour

Step 3: The train covers the remaining three quarters of the journey at a speed of 75 miles/hour.

Step 4: The time taken to cover the remaining three quarters of the journey is given by: Time = Distance/Speed
Time = $\frac{3}{4} \times \frac{75}{100}$ Time = $\frac{3}{4}$ hour

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.

Total Time = $\frac{1}{4} + \frac{3}{4}$ Total Time = $\frac{4}{4}$ Total Time = 1 hour

Therefore, the total duration of the journey is 1 hour.

LM²

Required Concepts:

Kinematics
Average Speed
Distance formula
Time formula

Relevant Equations:

Average Speed = Total Distance / Total Time
Distance = Speed $\times$ Time
Time = Distance / Speed

Subquestions:

- What is the total distance traveled by the train?
- How much time does it take for the train to cover the first quarter of the distance?
- What is the remaining distance to be covered after the train slows down?
- How much time does it take for the train to cover the remaining distance at the reduced speed?
- What is the total journey duration?

Subanswers:

- Total distance = Initial speed x Total time = 100 miles/hour x 10 hours = 1000 miles
- Time taken to cover the first quarter of the distance = Distance / Speed = 1000 miles / 4 / 100 miles/hour = 2.5 hours
- Remaining distance after covering the first quarter = Total distance - Distance covered = 1000 miles - 250 miles = 750 miles
- Time taken to cover the remaining distance at reduced speed = Remaining distance / Reduced speed = 750 miles / 75 miles/hour = 10 hours
- Total journey duration = Time taken to cover first quarter + Time taken to cover remaining distance = 2.5 hours + 10 hours = 12.5 hours

Final Answer: The total journey duration is 12.5 hours.

Figure 3: Comparison of GPT-4, DaSLaM and LM² on an example from the MATH dataset.

Dataset	Mixtral	Mixtral LM2	GPT 3.5	GPT 3.5 LM2
P-ALG	14	21	51	61
NT	40	46	26	41
PnC	20	30	23	31
Calc	15	19	18	21

Table 3: Performance of LM² on different solver models

Dataset	GPT-3.5 D+V	LM2	Frozen Llama-2 13B V
P-ALG	58	65	50
NT	38	43	22
PnC	30	34	15
Calc.	20	23	14

Table 4: Performance Comparison of LM² with frozen model baselines where D represents a frozen decomposer and V is a frozen verifier

fier modules: To study the effect of finetuning the decomposer and verifier modules as compared to utilising pretrained models for these purposes, we perform experiments using pretrained GPT 3.5 as decomposer/verifier and frozen Llama-2-13B verifier. From Table 4 it can be seen that indeed, a decomposer and a verifier with finetuning demonstrate the superiority achieved without finetuning.

Example analysis. To further understand the nuances of LM², we perform an analysis of the generated output on an example from the MATH dataset (see Figure 3). We compare between LM², 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 $\frac{\frac{3d}{4}}{75}$ 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 questions, the same problem is asked to be solved and the solver continues to make incorrect responses. With LM^2, we observe a much more well-defined, crisp line of questioning by the decomposer model; the solver is able to reach the correct answer option without regenerating the same information or drawing incorrect subanswers.

6 Conclusion

In this paper, we present LM^2, a cooperative cohort of generative language models working together to solve complex reasoning problems. LM^2 utilizes a frozen solver model that is guided to solve reasoning 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^2 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 reasoning examples, our proposed structured response scheme along with the fine-grained verification strategy plays a crucial role in generalizing LM^2 to heavily out-of-distribution tasks like medical question answering and chemistry.

Limitations

Despite promising results, LM^2 bears some inherent limitations. Compared to purely prompting-based methods, it requires a certain computational overhead for the two-staged training. With proprietary LLM-based solvers, LM^2 incurs extra token usage over single-pass solutions like chain-of-thought. Implicit limitations of the solver model, like lack of length generalization, arbitrary digit manipulation, etc. are expected to be inherited in LM^2 as well. A possible future work can be towards incorporating deterministic solvers and tools into the multiplex.

Ethical considerations

The research results are communicated honestly and credibly and transparency has been maintained throughout the research process. Innovations concerning automated problem-solving bear important ramifications in terms of human labor-value. This research is no exception in terms of future application. This research relies on the usage of proprietary models for building datasets and running experiments. The inequality in accessibility of such resources among research communities might affect future usage of this work.

Acknowledgments

The authors would like to acknowledge the financial support of DYSL-AI.

References

Daman Arora, Himanshu Gaurav Singh, and Mausam. 2023. Have llms advanced enough? a challenging problem solving benchmark for large language models. *Preprint*, arXiv:2305.15074.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Subhabrata Dutta, Ishan Pandey, Joykirat Singh, Sunny Manchanda, Soumen Chakrabarti, and Tanmoy Chakraborty. 2024. Frugal lms trained to invoke symbolic solvers achieve parameter-efficient arithmetic reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17951–17959.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Di Jin, Eileen Pan, Nassim Oufattole, Wei-Hung Weng, Hanyi Fang, and Peter Szolovits. 2020. What disease does this patient have? a large-scale open domain question answering dataset from medical exams. *Preprint*, arXiv:2009.13081.

Gurusha Juneja, Subhabrata Dutta, Soumen Chakrabarti, Sunny Manchanda, and Tanmoy Chakraborty. 2023. Small language models fine-tuned to coordinate larger language models improve complex reasoning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3675–3691, Singapore. Association for Computational Linguistics.

Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. 2022. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp. *arXiv preprint arXiv:2212.14024*.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2023. Making large language models better reasoners with step-aware verifier. *Preprint*, arXiv:2206.02336.

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

OpenAI. 2023. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Y Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Hao Sun, Hengyi Cai, Bo Wang, Yingyan Hou, Xiaochi Wei, Shuaiqiang Wang, Yan Zhang, and Dawei Yin. 2023. Towards verifiable text generation with evolving memory and self-reflection. *arXiv preprint arXiv:2312.09075*.

Denis Tarasov and Kumar Shridhar. 2024. Distilling llms' decomposition abilities into compact language models. *Preprint*, arXiv:2402.01812.

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

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. 2023. Large language models are better reasoners with self-verification. *Preprint*, arXiv:2212.09561.

Zhuofeng Wu, He Bai, Aonan Zhang, Jiatao Gu, VG Vinod Vydiswaran, Navdeep Jaitly, and Yizhe Zhang. 2024. Divide-or-conquer? which part should you distill your llm? *Preprint*, arXiv:2402.15000.

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

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

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.

A Training Details and Evaluation

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

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

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

B Training Data Creation

The data was generated using GPT-4. A temperature of 0.7 is used to ensure diversity in the generated data. We only stored the sub-question, sub-answer dataset if the number of sub-questions generated was more than three, this was done to ensure high data quality so that the model is able to decompose longer and more difficult questions effectively. First, we generate all the concepts, then the sub-questions given the question and the gold chain of thought. Finally, we generate the sub-answer given the question, a gold chain of thought and the sub-question to be answered. for the verifier, we first ask the LLM to answer the given question using standard COT prompting. Then based on the correctness of the answer, we take the solution chain of thought produced by the LLM and the gold answer and ask the LLM to classify the produced solution based on the mistake made. If the answer is correct, we store it separately and include it to make up to 10% of the dataset with the label as 'No Mistake'. Prompts for the data curation are given below.

B.1 Verifier Data Creation

B.1.1 Prompt

You are a teacher, and you are grading a student's answer to a question. The student's answer is as follows: {COT_LLM} The correct answer is as follows: {COT_gold} Please provide feedback to the students on the mistakes they have made. You need to fill out a rubric and classify the mistakes into the following categories:

1. Conceptual Mistakes: The student has misunderstood the concept or has applied the wrong concept.
2. Computational Mistakes: The student has made a mistake in the calculations.
3. Procedural Mistakes: The student has made a mistake in the procedure of solving the problem.
4. Mistake in understanding the question: The student has made a mistake in understanding the question.
5. Mistake in the first step: The student has made a mistake in the first step of the solution.
6. Mistake in the first half: The student has made a mistake in the first half of the solution.
7. Mistake in the second half: The student has made a mistake in the second half of the solution.
8. Mistake in the last step: The student has made a mistake in the last step of the solution.
9. No mistake: The student has not made any mistake.

Please first provide feedback then fill the rubric and then finally tell your feedback to the student in between <feedback> and </feedback> tags as shown below:
For example, if you want to tell the student that they have made a mistake in the first step and a conceptual mistake, then you need to write the following:
<feedback> 1,4 </feedback> Do not write anything else in between <feedback> and </feedback> tags except the numbers.

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

B.2 Decomposer Data Creation

B.2.1 Concepts data creation

I have a question's solution, tell me all the specific concepts, theorems and formulas (separated by a comma,) used in 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 $\boxed{0}$ prime numbers in the given row.
Concepts: Coefficients in Pascal's Triangle, Binomial Coefficients Formula, Prime Numbers
Question: question
Answer: answer
Concepts:

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 $\boxed{0}$ prime numbers in the given row.
Sub-questions:
$ question(How can the first two numbers be represented in form of binomial coefficients?)$, $ question(What are the values of all the coefficients in the row?)$, $ question(How many of the above numbers are prime?)$
Question: question
Answer: answer
Sub-questions:

B.3 Sub-answer data generation

I have a question, it's solution and a sub-question.
I want you to answer the subquestion along with an explanation.
Make sure to put the sub-answer in the answer tag like $sub-answer(The acceleration of the car at time t = 2 seconds is speed / time = 2m/s/2s = $1m/s^2$)$
Think step by step.
Question: question
Answer: answer
Sub-question: sub-question-array[i]
Sub-Answer: