TEACHING ALGORITHMIC REASONING VIA IN-CONTEXT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have shown increasing in-context learning capabilities through scaling up model and data size. Despite this progress, LLMs are still unable to solve algorithmic reasoning problems. While providing a rationale with the final answer has led to further improvements in multi-step reasoning problems, Anil et al. (2022) showed that even simple algorithmic reasoning tasks such as parity are far from solved. In this work, we identify and study four key stages for successfully teaching algorithmic reasoning to LLMs: (1) formulating algorithms as skills, (2) teaching multiple skills simultaneously (skill accumulation), (3) teaching how to combine skills (skill composition) and (4) teaching how to use skills as tools. We show that it is possible to teach algorithmic reasoning to LLMs via in-context learning, which we refer to as *algorithmic prompting*. We evaluate our approach on a variety of arithmetic and quantitative reasoning tasks, and demonstrate significant boosts in performance over existing prompting techniques. In particular, for long parity, addition, multiplication and subtraction and parity tasks, we achieve an error reduction of approximately 10x, 9x, 5x and 2x respectively compared to the best available baselines.

1 INTRODUCTION

Large language models (LLMs) have shown impressive progress in recent years, driven by the scaling up of models and training data sizes (Kaplan et al., 2020; Wei et al., 2022a; Hoffmann et al., 2022) that has led to improved performance and sample efficiency (Brown et al., 2020; Chen et al., 2021; Chowdhery et al., 2022). One area with significant room for improvement is the ability of LLMs to perform complex reasoning tasks. In this realm, mathematical reasoning (Saxton et al., 2019) provides a unique challenge as a domain. It requires the ability to parse, to logically deconstruct a problem into sub-problems and recombine them, and to apply knowledge of rules, transformations, processes, and axioms.

The idea of providing a rationale with the final answer was first proposed by Ling et al. (2017) and recently revived for LLMs in the form of scratchpad (Nye et al., 2021) and chain-of-thought (Wei et al., 2022b). It has led to improvements in performance on multi-step reasoning problems (Wang et al., 2019) such as arithmetic, commonsense, and symbolic reasoning tasks (Nye et al., 2021; Wei et al., 2022b; Lewkowycz et al., 2022a; Wang et al., 2022a;b; Anil et al., 2022; Zhou et al., 2022). However, despite significant progress, these models still struggle with out-of-distribution (OOD) generalization on reasoning tasks (Nogueira et al., 2021; Kim et al., 2021; Anil et al., 2022).

To successfully generalize out-of-distribution on many of these reasoning tasks, the model needs to learn the underlying algorithm for solving a task. We refer to this behavior as *algorithmic reasoning* (Kaiser & Sutskever, 2015; Veličković & Blundell, 2021). While following an algorithm can be seen as a form of instruction following, algorithms are generally more complex with a larger number of steps, although each algorithm step may be simpler and more concise than typical instructions. The benefit of being able to learn algorithms is that since they are input independent by nature, they guarantee good OOD performance when executed properly. Moreover, algorithms can be specified without ambiguity and hence provide a good test bed to probe model capabilities.

One surprising capability of LLMs is in-context learning (Brown et al., 2020), which refers to the ability to learn a task from a few examples being presented within a *prompt*. In-context learning does not require any weight updates, and provides a powerful platform for specialized skill acquisition

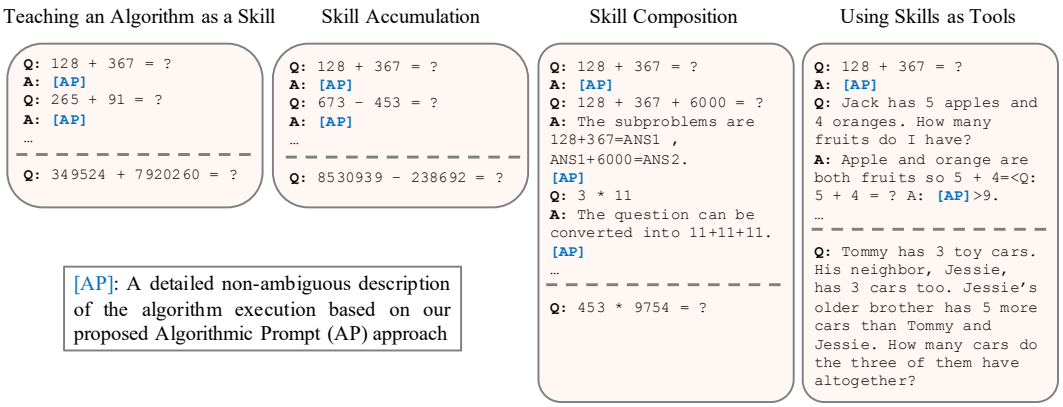


Figure 1: The four learning stages investigated in this work (from left to right): (i) Teaching an algorithm as a skill (Section 3) (ii) Skill Accumulation, i.e., teaching multiple skills simultaneously (Section 4) (iii) Skill Composition, i.e. the ability to learn a complex skill through simpler ones (Section 5) (iv) Using Skills as Tools to solve problems (Section 6).

without losing the generality of the underlying model. Moreover, various prompting strategies have shown significant potential in solving certain types of reasoning problems (Jung et al., 2022; Zhou et al., 2022; Wei et al., 2022b; Kojima et al., 2022). Nonetheless, Anil et al. (2022) considered two algorithmic reasoning tasks and showed that while rationale-based prompting allow LLMs to generalize to longer problem instances, they are still far from solving simple tasks such as parity.

In this work, we investigate how to teach algorithms and compositions of algorithms to LLMs via in-context learning. This setup is reminiscent of how similar skills are taught to children in school. We identify and explore four key stages for teaching algorithms as skills to LLMs (Figure 1). We begin by studying the shortcomings of existing approaches and propose ways to alleviate them. We focus on arithmetic algorithms such as addition, subtraction and multiplication as they have been widely benchmarked (Saxton et al., 2019; Hendrycks et al., 2021) and famously fail at out-of-distribution generalization even for best performing models on the MATH benchmark (Lewkowycz et al., 2022b). While one can avoid learning these algorithms by the use of external tools (Cobbe et al., 2021b), this approach cannot scale to higher levels of abstraction where a model needs to use "soft algorithms" and certain steps must be flexibly applied in different situations.

Contributions: Our main contributions are as follows:

- We introduce *algorithmic prompting*, which involves providing a detailed description of the algorithm execution on running examples, and using explicit explanation and natural language instruction to remove ambiguity. For a comparison of algorithmic prompting to existing prompting techniques, see Section 2 and Table 1.
- We demonstrate that algorithmic prompting significantly outperforms existing prompting techniques on several algorithmic tasks. In particular, for long parity, addition, multiplication and subtraction and parity tasks, we achieve an error reduction of approximately 10x, 9x, 5x and 2x respectively compared to the best available baselines (Section 3 and Table 2).
- Our ablation studies reveal the impact of non-ambiguous explanations, and show that unlike other prompting approaches, errors in the algorithmic examples affect performance significantly.
- We study the model's ability to simultaneously learn multiple algorithms via a single prompt, as well as its ability to compose the learned algorithms in order to solve more complex tasks (Sections 4 and 5).
- We explore several potential approaches to leverage a learned algorithm as a tool to solve math word problems. We show that while it is possible to improve the performance in settings that required complex calculations, the model's general reasoning capability reduces due to the phenomenon of *interference* (Section 6).

2 Algorithmic Prompting

The idea of getting the model to show its work, i.e, breaking the problem down and asking the model to output the intermediate steps used in solving the task was proposed in Nye et al. (2021).

The authors show that by finetuning LLMs on such data – which they refer to as *scratchpad* – they can greatly improve performance on multi-step computation problems. This was taken further by Wei et al. (2022b) to the in-context learning setting, where they showed that providing rationales in the prompts significantly increases the model's ability to solve similar reasoning problems. They refer to this approach as *chain-of-thought*. The main intuition behind the scratchpad approach is that by having intermediate computations in the output, the model can refer to them directly instead of relying on its internal representation space for those calculations. For chain-of-thought, one hypothesis is that the rationales loosely provide the model with a "thinking pattern" that it can reference when tackling a problem. By encouraging the model to output an explanation along with the answer, we steer it towards solving problems by breaking them into steps that logically follow from each other. Inspired by these perspectives, we hypothesize that if we increase the specificity and applicability of these thinking patterns, we can also increase the amount by which the model adheres to these patterns in its problem solving. As we will illustrate, this approach leverages both the scratchpad ideas of showing intermediate computations and the chain-of-thought ideas of providing an explanation for each step.

As a motivating example, consider the standard addition algorithm. This method right-aligns the two numbers being added and calculates the sum of pairs of single digits from each number, going from right to left. For every pair of digits, there is a possible carry that needs to be added to the next digit sum. If we use a scratchpad-style illustration, then for a question like $182 + 376$, the model would see that the digit-sum $2+6$ generates a carry of 0, while $8+7$ generates a carry of 1. However, from just this example the rules of carry is highly ambiguous. Ideally, we expect the model to conclude that when $a + b > 9$, it generates a carry of 1, and when $a + b \leq 9$, it generates a carry of 0. But from the scratchpad-style example the model could have concluded that the carry is 1 whenever we add two even digits together and 0 otherwise, or that the first digit-pair generates a carry of 1, the second digit-pair generates a carry of 0, and so on. In order for the model to extrapolate the correct pattern, it must be biased in such a way that the general and correct rule is the default interpretation. Such alignment, however, can not be reliably expected from current models.

We hypothesize that existing prompting methods fail to sufficiently constrain the model's interpretation of the prompting information, and result in unexpected and undesirable model behaviors on tasks that require precise algorithmic reasoning. We push the limits of rationale-based prompting by drastically increasing the amount of detail included in the rationales, while specifying the steps of an algorithm within this information. We refer to this strategy as *Algorithmic Prompting*. We show that it can achieve significant systematic generalization on the several algorithmic reasoning tasks, and ground our exploration in the four capabilities identified in Figure 1.

Table 1: Comparison of different prompting strategies studied in this work. The number of ★ indicates the level to which each strategy exhibit the given characteristic. We see that algorithmic prompt includes both qualities of natural language explanation and explicit intermediate computations.

Prompt Strategy	**Input-target pairs**	**Natural language rationale**	**Intermediate computations**	**Multiple rationales possible**
Few-shot	★★★	-	-	-
Chain-of-thought	★★★	★★★	★	★★★
Scratchpad	★★★	-	★★	-
Instruction-only	-	★★★	-	★★★
Algorithmic	★★★	★★	★★★	★

3 TEACHING ALGORITHMS AS SKILLS

Baselines: We mainly compare the proposed algorithmic prompt to few-shot and chain-of-thought baselines in our experiments. The *few-shot* baseline refers to the simple approach of presenting examples of question and answer pairs with no additional explanation. The *chain-of-thought* baseline provides a rationale along with the final answer in the few-shot examples. In order to generate the rationale for various tasks, we follow the method introduced in Kojima et al. (2022) and use the phrase "let's think step by step" to get a model-generated rationale for the few-shot examples.

Evaluation Metric: We measure both in-distribution and OOD performance in all experiments. In particular, we use *length generalization*, where the model sees shorter examples at training time,

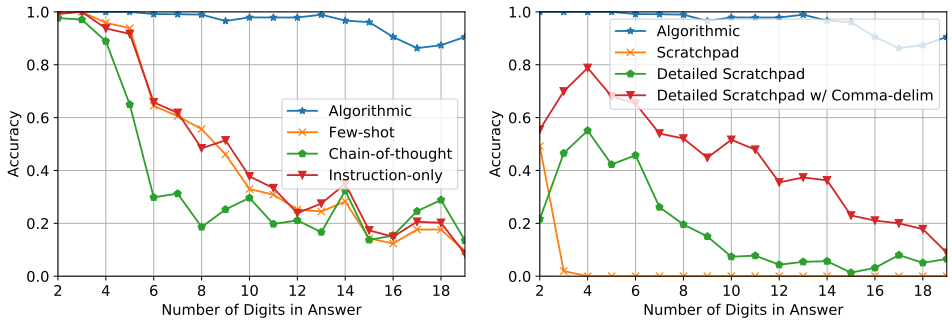


Figure 2: Accuracy on addition questions of increasing length for different prompting methods. Addition questions are of the form $a + b = c$, where a, b, and c are positive integers. The number of digits in answer plotted in the x-axis refers to the length of c. Accuracy is measured over 2000 total examples sampled uniformly over the length of c. The max length for examples in the prompt is 5. **Left:** We see that algorithmic prompt shows near-perfect length generalization even on extremely long addition questions, and significantly outperforms its simple few-shot and chain-of-thought counterpart. **Right:** using scratchpad-style output as a prompt leads to abysmal performance, but adding a few extra details to the scratchpad format leads to non-trivial generalization.

and generalize to longer problems at test time. Length generalization provides a good indication for whether the model has learned the underlying algorithm.

Experimental Setting: For all the experiments in this section and in the rest of the paper, we use the Codex model (Chen et al., 2021) `code-davinci-002` from OpenAI. This model has a maximum context length of 8000 tokens. Task examples are sampled uniformly at each length. All results are sampled once using a temperature of 0 and default settings for other parameters.

3.1 Two-number addition

We begin our analysis by studying the two-number addition task and explore the effectiveness of various prompting strategies, going from most ambiguous to least ambiguous. The addition problem takes the form $a + b = c$ where a, b, and c are positive integers.

We present an algorithmic prompt for addition and compare its performance against the few-shot, chain-of-thought, instruction-only, and scratchpad methods. An illustration of these prompting strategies for addition is shown in Figure 14, and the prompt used for instruction-only is shown in Section G.5. In addition, we experiment with a more detailed version of the scratchpad format (referred to as *detailed scratchpad* herein) to see how adding details about intermediate steps changes the performance of an existing prompt in prior work. We also consider converting the numbers from space-delimited to comma-delimited (*detailed scratchpad w/ comma-delim*). For all addition experiments, we restrict prompt examples to answers up to 5 digits in length, and evaluate on questions up to 19 digits in length. The length of 19 is chosen because this is the level after which the model begins to run out of context. A similar choice is used for all algorithms considered in Section 3.

Figure 2(a) shows the performance of algorithmic prompt against existing methods on addition problems. These results demonstrate that algorithmic prompt achieves near perfect performance and OOD generalization on addition, while few-shot, chain-of-thought, and instruction-only have decreasing performance as the length of the answer increases. Interestingly, chain-of-thought prompt did not improve performance over simple few-shot, suggesting that insufficient rationales may even hinder the model's natural abilities. These results illustrate the benefit of incorporating algorithmic steps, unambiguous explanations, and demonstrations on running examples in our prompt.

Figure 2(b) compares the performance of scratchpad and detailed scratchpad. We conjecture that the abysmal performance of scratchpad as a few-shot prompt is due to the structure of the solution format being sufficiently regimented to move the model away from its memorized solutions, but not clear enough for the model to extract the true underlying rules and adapt them to new examples. With detailed scratchpad, we add more intermediate steps to illustrate how the values of the answer (A) and the carry (C) is derived. An addition version converts the numbers from space-delimited to comma-delimited, as we observed that comma is a more effective deliminator for Codex. We find that the additional information leads to a significant boost in results.

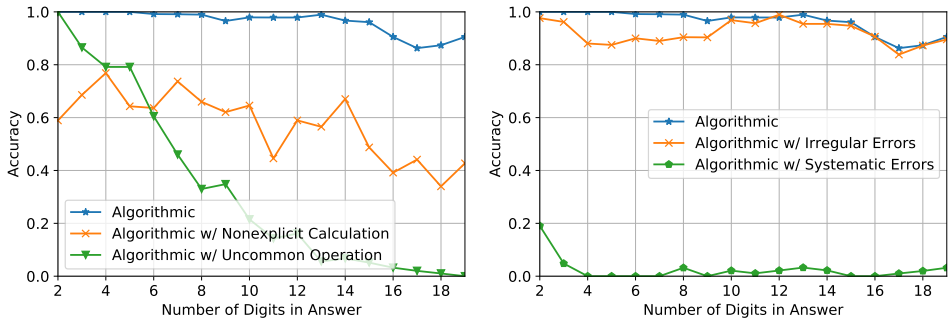


Figure 3: Accuracy on addition questions of increasing length for variants on the algorithmic prompt . **Left:** Two examples of rule ambiguity that we address in the final addition prompt are non-explicit carry calculation (*Nonexplicit Calculation*) and digit indexing (*Uncommon Operation*). We observe a significant difference in performance before and after reducing the ambiguity of these operations. **Right:** Errors are introduced to the algorithmic prompt in the digit value of the second number in the equation. *Sampled errors* are introduced to a minority subset of steps in the algorithmic examples, while *systematic errors* are introduced to all steps of the examples. We see that sampled errors have a minor impact on the performance, and systematic errors completely destroy the model's ability to solve this task. This suggests that the model is following the algorithm as it is specified in-context, rather than loosely mimicking the format of the algorithm.

We compare the algorithmic prompt to two less-detailed variants. One version (*nonexplicit calculation*) omits the explicit equation showing how the carry value is derived. This shares the same intuition as the original motivating example. The second version (*uncommon operation*) requires the model to index the correct digit for a given step. The indexing of a digit at a variable position is a more uncommon operation than the indexing of the digit at the same position each time. In our final addition prompt, we introduce a mechanism that allows the model to avoid the indexing operation by copying the unprocessed digits over to each step and always taking the last digit. See an example of this in Section G.2. Figure 3(a) illustrates the relative gains that come from the disambiguation of these two aspects of the algorithm.

Is the model actually learning the algorithm through in-context learning? Min et al. (2022) have shown that it is not necessary to provide the correct question-answer pairings in the few-shot prompt, suggesting that the model does not rely on the demonstrations themselves to figure out the right way to solve the given task. However, in order to claim that we are teaching algorithms in-context, we would like to understand whether the model is actually following the algorithm as it is prescribed in the prompt. To do so, we validate that 1) mistakes in the intermediate output steps lead to mistakes in the final answer, and 2) errors in the prompt significantly impact performance.

We first look at the errors that the model makes. We find that for every addition question where the final answer was correct, *all* intermediate steps were also correct. Next, we analyze the performance of the model when we introduce errors into the algorithmic steps in the prompt. We introduce errors into second digit of the calculation step ($\text{digit}_1 + \textit{digit}_2 + \text{carry} = \text{answer}$), and keep all other elements the same as before. We consider two types of errors: *irregular errors* where only a subset of the steps contain an error, and *systematic errors* where all of the steps presented in the prompt contain an error. With irregular errors (Section G.3), the model still has a chance of extrapolating the correct rule based on the unchanged steps. With systematic errors (Section G.4), the model should *not* derive the correct rule if it was truly learning from context, rather than simply mapping to the output format and overriding the individual steps with what it has learned from its pretraining. Figure 3(b) shows that there is a small degradation in performance with irregular errors, whereas the accuracy drops to near 0% with systematic errors, thus confirming the expected behavior of a model that is actually learning in-context. This is in contrast to the findings in which providing shuffled targets (Min et al., 2022) or wrong patterns in chain-of-thought (Madaan & Yazdanbakhsh, 2022) do not materially impact model's performance. Thus, algorithmic prompting differs from other approaches and constrains the model's behavior towards what is being taught in-context.

3.2 TEACHING OTHER ALGORITHMS USING ALGORITHMIC PROMPT

To validate that the performance of algorithmic prompt is not specific to two-number addition, we evaluate model performance on three other algorithms: subtraction, multiplication, and parity.

Subtraction: We follow a similar strategy as addition for subtraction. We discuss the peculiarities of the subtraction algorithm in more detail in Section 4, where we combine both addition and subtraction problems. The results are summarized in Table 2.

Multiplication: For multiplication, we consider questions in the form of $a \times b = c$. Multiplication requires $O(n^2)$ steps if we use a similar strategy as the addition algorithm (which takes $O(n)$ steps). Inspired by this complication, we use multiplication to explore whether the model's existing zero-shot or few-shot capabilities can be leveraged in conjunction with algorithmic prompting to reduce the complexity of the required instructions. Instead of using single-digit multiplication in each step, we perform direct calculations for single-digit $\times$ n-digit numbers. Instead of doing n^2 single-digit calculations for two n-digit numbers, we now only need to perform n steps of $1\times$n-digit multiplication. For more details of the algorithm and results, see Figure 4 and Section G.7. We see in Table 2 that the multiplication algorithm performs well compared to its few-shot counterpart. These results point to the potential of utilizing a model's inherent abilities within the scaffolding of more structured algorithmic instructions.

Parity: Next, we consider the problem of teaching the model to calculate parity of a given binary list. This task has been studied extensively in Anil et al. (2022), and despite the intrinsic simplicity of the algorithm, it is far from being solved. We see in Table 2 that algorithmic prompt significantly outperforms random chance on this task, which is even greater than the few-shot prompting performance reported in Anil et al. (2022). More details can be found in Figure 9 and Section C.

Table 2: Performance on subtraction, multiplication, and parity tasks. For addition we use the chain-of-thought baseline and evaluate at length 19. For subtraction we use the few-shot baseline and evaluate at length 15. For subtraction we use the few-shot baseline and evaluate at length 8. These lengths are chosen based on the maximum task length that could fit into context. For parity we evaluate at length 20 which is the longest instance reported in Anil et al. (2022).

Method	Addition	Subtraction	Multiplication	Parity
Algorithmic prompt	90.5%	65.6%	79.2%	95.0%
Best available baseline	13.5%	16.7%	2.9%	50.0%

4 SKILL ACCUMULATION

So far we have demonstrated the ability to teach single-algorithms through in-context learning. In this section, we study the model's ability to simultaneously learn multiple algorithms and choose the applicable one when solving problems. To do so, we use the addition-subtraction task. We expand on the addition problem to allow for both positive and negative numbers. Thus, the problems now have four possibilities: $a + b$, $-a + b$, $-a - b$, $a - b$. We refer to questions of the form $a + b$ as *addition-only* questions, and the rest as *subtraction-only* questions. For subtraction questions, the ordering of the two numbers matter. To see this, consider the examples $43 - 250 = -207$ and $543 - 250 = 293$. When we process the digits from right to left, the answer depends on whether the first number is greater than or less than the second number in absolute value. Thus, subtraction requires a different – albeit similar – algorithm to addition. For a sense of the relative complexity of the two settings, note that the subtraction algorithm we use runs in $2n$ steps, while the addition algorithm runs in n steps.

To succeed at this task, the model needs to demonstrate the ability to follow different processing paths when the question is addition or subtraction. Figure 5 shows the performance of the combined addition-subtraction prompt, with the accuracy broken down by question type. We see that the model is able to effectively execute the correct algorithm based on the individual questions. The model also exhibits lower accuracy on subtraction questions compared to addition-only questions, reflecting the increased complexity of the subtraction algorithm. Comparing the performance on addition-only questions to the addition prompt from Section 3.1, we see that there is minimal change in performance despite having an extra other algorithm present in the prompt.

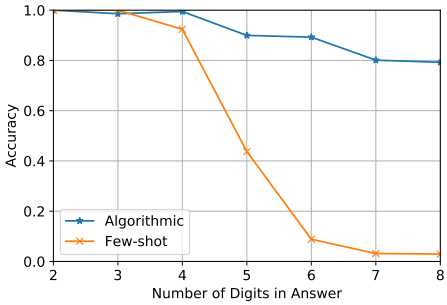


Figure 4: Performance of algorithmic prompt on multiplication questions, where at least one of the two numbers in the question has 3 or fewer digits. Algorithmic prompt shows superior length generalization compared to the few-shot baseline.

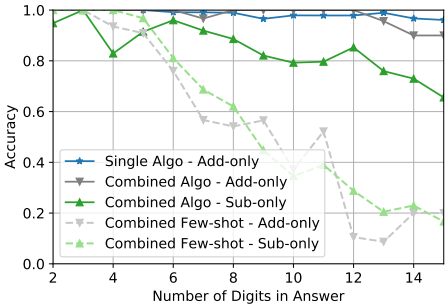


Figure 5: Accuracy on addition and subtraction questions using a combined prompt. There is minimal degradation in performance for addition-only questions using the combined prompt compared to the single algorithm addition-only prompt.

5 Skill Composition

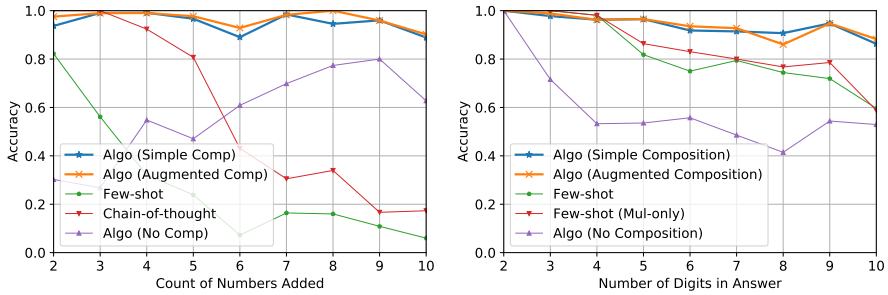
In this section, we explore the model's ability to learn multiple algorithms that build on top of each other. This is a desirable property because it enables the model to learn a more complex algorithm without having to relearn simpler sub-components of that algorithm. This setting allows for the accumulation of skills and the modularization of complex algorithms. To establish a framework for skill composition, we explore two extensions to the addition algorithm: 1) adding multiple numbers together, and 2) solving multiplication as an addition problem (e.g. by converting $3*7$ into $7+7+7$). The ability to add multiple numbers builds on top of the ability to add two numbers together, which we already established. Solving multiplication as addition further builds on the addition of multiple numbers. An illustration can be found in Figure 11.

We teach these algorithms in-context by creating a *combined prompt* that includes 2 examples from the addition prompt, 1 example of addition of 3 numbers, and 1 example of converting multiplication into addition. This forms a simple composition strategy (*Algo - (Simple Comp)*). We also consider two ablations of the combined algorithmic prompt. The algorithm for n-number addition involves wrapping 2-number additions within a larger loop of $n-1$ addition problems. Thus, we could provide even more information by converting the 2-number addition prompt examples into the same loop format as the 3-number addition example. This version (*Algo - (Augmented Comp)*) provides an upper bound to multi-number addition and multiplication-as-addition. The second ablation (*Algo - (No Comp)*) only presents the example that illustrates the extended skill. This has no composition and provides a lower bound for the two extended skills, and illustrates the improvement that comes from having first learned the component algorithms.

In-context skill composition is limited by the context length of current models. Unlike the previous experimental results, these composition tasks include a number of questions that were incomplete for the algorithmic prompt . To separate out the issue of context length from the ability of the model to follow an algorithm, we report performance in Figure 6 on only the questions for which the algorithmic prompt could fit into context. This subset is also used for all baselines. In order to move past context length limitations, we further experiment with a second-pass strategy, where we keep only the last completed algorithmic step in the model's output, and perform a second inference pass using the original prompt and the last output step. This simple approach benefits from the fact that all relevant state variables are outputted in each step of the algorithm. We report performance on the entire dataset using the second pass strategy in Figure 12. Figure 6 shows that the algorithmic prompt significantly outperforms few-shot and chain-of-thought baselines. Moreover, we observe that there is minimal difference between the simple composition and augmented composition strategies, and that the no composition approach performs much worse than its composed counterparts.

6 Using skills as tools

The addition and subtraction algorithms are similar in nature and format, and may thus be more synergistic than other combinations. In this section, we study the behavior of the model when using a given algorithm as a step in solving a larger reasoning problem. To do so, we leverage the



(a) Multi-number addition

(b) Multiplication-as-addition

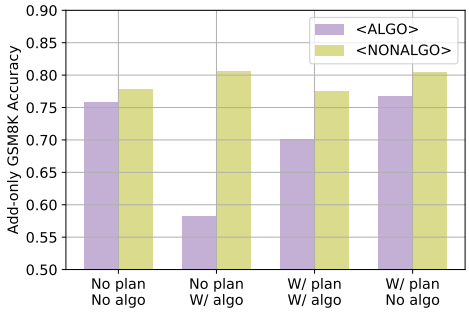
Figure 6: Performance on compositions of skills. "Algo" indicates algorithmic prompting. "Simple Comp" refers to a simple composition strategy where previously taught algorithms are transferred as is. "Augmented Comp" adjusts the previously taught algorithm to match the format of the new task. This simulates a version where the full prompt specializes to the new task. "No Comp" uses only the part of the "Simple Comp" prompt that describes the new task. This simulates the comparison to learning a new skill from scratch without first learning its stepping stones. We observe that the composed algorithmic templates demonstrate better generalization than the baseline methods. Note that for multiplication, we evaluate the algorithmic methods on a harder task than the few-shot baselines, since we force the model to convert the question into the addition of a number n times, while for other baselines we simply perform $1 \times n$-digit multiplication directly.

grade school math word dataset (GSM8k), which consists of high-quality mathematical reasoning problems presented as natural language questions (Cobbe et al., 2021a).

GSM8k questions consist of two components: 1) the informal mathematical reasoning component which requires the model to come up with the correct solutions steps to arrive at the answer based on the information provided in the question, and 2) the calculation of basic arithmetic operations used in the solution steps. In this section, we use GSM8k as a test bed for studying how the model can leverage a learned algorithm (instead of an external calculator) inside a broader reasoning process.

Augmenting informal mathematical reasoning: If we think of chain-of-thought prompting used in GSM8k as providing a soft pattern for mathematical reasoning to the model, then learning the addition algorithm in conjunction with mathematical reasoning is akin to the multi-skill settings previously. However, in this case, we not only need to ensure that both skills are learned effectively, but also that the mathematical reasoning skill be augmented to call on another algorithm when applicable. In order to study the ability to use the addition algorithm while solving GSM8k questions, we simplify the task by filtering for a subset of GSM8k whose solutions consist of only addition steps. The filtering procedure results in 108 pure-addition GSM8k questions. We use a few-shot prompt where a subset of the questions are augmented with algorithmic output (identified through a `<ALGO>` flag), while the remaining questions (identified through a `<NONALGO>` flag) are the same as what's used in chain-of-thought. For more details about this approach, see Section F. See Section G.9 for the actual prompt. Figure 7(a) shows the results using this approach. We start with the chain-of-thought baseline with the added flags ("No plan No Algo"), and compare it to the same prompt augmented with algorithmic output ("No plan W/ algo"). We see that including algorithmic output in the `<ALGO>` examples significantly disrupts the mathematical reasoning components of the solution while leaving the `<NONALGO>` performance relatively unchanged. We refer to this phenomenon as *interference*, and conjecture that this is particularly prominent due to the highly different nature and format of the mathematical reasoning and algorithmic addition skills. In order to help the model set up the correct mathematical reasoning steps before being distracted by algorithmic output, we augment the chain-of-thought portion of `<ALGO>` prompt examples with a solution plan at the beginning of the output. Although this alleviates some of the interference, the performance ("W/ plan W/ algo") is still much worse than the same version without algorithmic output ("W/ plan No algo"). Overall, we find that using flags in the prompt is a simple way of directing a model's attention paths, and that the composition of multiple skills inside the same context may face the issue of interference.

Algorithmic prompt as tool use: To illustrate the potential of skill composition as a form of tool use, we create a hard dataset using 50 examples from the pure-addition subset. This dataset increases the numerical values used in the questions, thus making the task more difficult for the model. An example from this dataset is presented in Figure 1 in "using skills as tools". Given the interference



(a) Performance on GSM8k addition-only subset

Q: June liked to collect cat stickers. She had a total of 768835 in her collection. Her twin sister Bonnie also collected cat stickers and had 63212920 in her collection. For their birthday, their grandparents gave them 250148 cat stickers each. How many combined stickers did they have?

A: June had <ALGO_START>768835+250148=<CALL_ADD_ALGO>1018983 stickers. Bonnie had <ALGO_START>63212920+250148=<CALL_ADD_ALGO>63463068 stickers. Together, they had <ALGO_START>1018983+63463068=<CALL_ADD_ALGO>64482051 stickers. The answer is 64482051.

(b) Example of tool use for GSM8k-Hard.

Figure 7: Addition algorithm as tool use in solving GSM8k questions. **Left:** Ablation study with or without algorithmic output in the prompt or output. We see that algorithmic output within-context leads to significant interference with the model's mathematical reasoning abilities. This is alleviated by using a summary before the algorithmic output, but not fully. **Right:** An example question-answer pair from the GSM8k-Hard addition dataset, which includes GSM8k-like questions with large numerical values.

Table 3: Performance on GSM8k-Hard addition dataset with or without algorithmic tool use.

Method	Overall Accuracy	Logical Accuracy	Addition Accuracy
Chain-of-thought w/ Algo call	55.8%	57.7%	98.4%
Chain-of-thought wo/ Algo call	27.4%	70.6%	61.9%

issue that we have identified, as well as the limitations on the context length, we introduce a way to side-step these issues through a dialogue-like interaction between models loaded with different prompts. In this case, we teach the model doing the mathematical reasoning steps to call on another model when it needs to perform addition. This involves teaching the model to output specific tokens to indicate when a separate model should be consulted. Then, we send the question to the second model loaded with the addition algorithmic prompt, which executes the addition algorithm and returns the answer back to the first model. The first model would then continue with the rest of the answer without suffering from having the algorithmic output in its context. Creswell & Shanahan (2022) uses a similar multi-model and multi-prompt in order to separate out selection from inference in reasoning problems. This can also be considered a form of tool use (Parisi et al., 2022), where a model queries another source for a particular type of information.

We see the performance on the GSM8k-Hard dataset in Table 3. We see that despite removing the algorithm output from the context of the first model, we still observe interference coming from the use of flags in the logical natural language solution steps. Nonetheless, the method that leverages the algorithmic tool use still achieves double the accuracy as the baseline method.

7 Conclusion and Future Work

Motivated by the potential of in-context learning as a general mechanism for compositional skill acquisition in LLMs, we studied teaching algorithmic reasoning via in context learning. We identified and studied the fundamental building blocks towards this goal and investigated four settings: teaching an algorithm as a skill, skill accumulation, skill composition and using skills as tools. We investigated the shortcomings of existing approaches and proposed algorithmic prompt to alleviate them, showing that it leads to significant performance boost in various algorithmic reasoning tasks.

We identified the interference phenomenon for tool use application and investigated different ways to reduce its effect. Our observations about interference suggest that teaching the model the ability to retrieve or selectively attend to specific instructions when solving the particular problem is an important future direction. Moreover, given that there are ongoing efforts in the community to increase the context length of LLMs, it is interesting to design more challenging tasks for each of the four introduced settings and investigate what capabilities can be thought to LLMs when having access to extremely large context length.

Reproducibility Statement: We have included the details of all the prompts in Appendix G. The LLM used in the paper is OpenAI's Codex which has free public access and hence anyone can reproduce our results. Moreover, GSM8k dataset is publicly available.

Ethics Statement: We believe that our submission raises no questions regarding the Code of Ethics as we use open public datasets and model. Our study also does not involve human subjects, potentially harmful insights, methodologies and applications, potential conflicts of interest and sponsorship, discrimination/bias/fairness concerns, privacy and security issues, legal compliance, and research integrity issues. We also referred to close related works and compare our results wherever applicable.

REFERENCES

Cem Anil, Yuhuai Wu, Anders Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. Exploring length generalization in large language models. *arXiv preprint arXiv:2207.04901*, 2022.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Xinyun Chen, Chen Liang, Adams Wei Yu, Dawn Song, and Denny Zhou. Compositional generalization via neural-symbolic stack machines. *Advances in Neural Information Processing Systems*, 33:1690–1701, 2020.

Ting-Rui Chiang and Yun-Nung Chen. Semantically-aligned equation generation for solving and reasoning math word problems. *arXiv preprint arXiv:1811.00720*, 2018.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

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. Training verifiers to solve math word problems. 2021a. URL `https://arxiv.org/abs/2110.14168`.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021b.

Antonia Creswell and Murray Shanahan. Faithful reasoning using large language models. *https://arxiv.org/abs/2208.14271*, 2022.

Keyur Faldu, Amit Sheth, Prashant Kikani, Manas Gaur, and Aditi Avasthi. Towards tractable mathematical reasoning: Challenges, strategies, and opportunities for solving math word problems. *arXiv preprint arXiv:2111.05364*, 2021.

Jonathan Gordon, David Lopez-Paz, Marco Baroni, and Diane Bouchacourt. Permutation equivariant models for compositional generalization in language. In *International Conference on Learning Representations*, 2019.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Albert Q Jiang, Wenda Li, Szymon Tworkowski, Konrad Czechowski, Tomasz Odrzygóźdź, Piotr Miłoś, Yuhuai Wu, and Mateja Jamnik. Thor: Wielding hammers to integrate language models and automated theorem provers. *arXiv preprint arXiv:2205.10893*, 2022.

Erik Jones and Jacob Steinhardt. Capturing failures of large language models via human cognitive biases. *arXiv preprint arXiv:2202.12299*, 2022.

Jaehun Jung, Lianhui Qin, Sean Welleck, Faeze Brahman, Chandra Bhagavatula, Ronan Le Bras, and Yejin Choi. Maieutic prompting: Logically consistent reasoning with recursive explanations, 2022.

Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Jeonghwan Kim, Giwon Hong, Kyung-min Kim, Junmo Kang, and Sung-Hyon Myaeng. Have you seen that number? investigating extrapolation in question answering models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7031–7037, 2021.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. 2022. URL `https://openreview.net/forum?id=6p3AuaHAFiN`.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*, 2022a.

Aitor Lewkowycz, Anders Andreassen, David Martin Dohan, Ethan S Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. 2022b. URL `https://arxiv.org/abs/2206.14858`.

Wenda Li, Lei Yu, Yuhuai Wu, and Lawrence C Paulson. Isarstep: a benchmark for high-level mathematical reasoning. *arXiv preprint arXiv:2006.09265*, 2020.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *ACL*, 2017.

Aman Madaan and Amir Yazdanbakhsh. Text and patterns: For effective chain of thought, it takes two to tango. *arXiv preprint arXiv:2209.07686*, 2022.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? *arXiv preprint arXiv:2202.12837*, 2022.

Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of transformers with simple arithmetic tasks. *arXiv preprint arXiv:2102.13019*, 2021.

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

Aaron Parisi, Yao Zhao, and Noah Fiedel. Talm: Tool augmented language models. *arXiv preprint arXiv:2205.12255*, 2022.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

Markus N Rabe, Dennis Lee, Kshitij Bansal, and Christian Szegedy. Mathematical reasoning via self-supervised skip-tree training. *arXiv preprint arXiv:2006.04757*, 2020.

Yasaman Razeghi, Robert L Logan IV, Matt Gardner, and Sameer Singh. Impact of pretraining term frequencies on few-shot reasoning. *arXiv preprint arXiv:2202.07206*, 2022.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*, 2019.

Avijit Thawani, Jay Pujara, Pedro A Szekely, and Filip Ilievski. Representing numbers in nlp: a survey and a vision. *arXiv preprint arXiv:2103.13136*, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Petar Veličković and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7):100273, 2021.

Haoyu Wang, Mo Yu, Xiaoxiao Guo, Rajarshi Das, Wenhan Xiong, and Tian Gao. Do multi-hop readers dream of reasoning chains? *arXiv preprint arXiv:1910.14520*, 2019.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. Rationale-augmented ensembles in language models. *arXiv preprint arXiv:2207.00747*, 2022a.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022b.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*, 2022a.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022b.

Sean Welleck, Jiacheng Liu, Ronan Le Bras, Hannaneh Hajishirzi, Yejin Choi, and Kyunghyun Cho. Naturalproofs: Mathematical theorem proving in natural language. *arXiv preprint arXiv:2104.01112*, 2021.

Yuhuai Wu, Albert Q Jiang, Wenda Li, Markus N Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. *arXiv preprint arXiv:2205.12615*, 2022.

Louis-Pascal Xhonneux, Andreea-Ioana Deac, Petar Veličković, and Jian Tang. How to transfer algorithmic reasoning knowledge to learn new algorithms? *Advances in Neural Information Processing Systems*, 34:19500–19512, 2021.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? *arXiv preprint arXiv:1905.13211*, 2019.

Yujun Yan, Kevin Swersky, Danai Koutra, Parthasarathy Ranganathan, and Milad Hashemi. Neural execution engines: Learning to execute subroutines. *Advances in Neural Information Processing Systems*, 33:17298–17308, 2020.

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

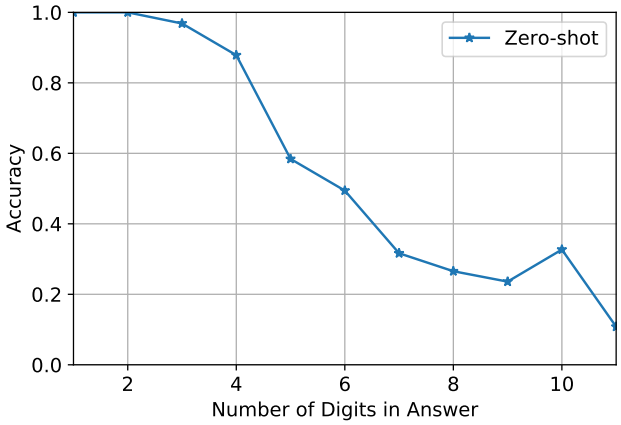


Figure 8: Zero-shot accuracy of one-digit multiplication, where the other number can have up to 11 digits. We see that the accuracy starts to drop after 3 digits in the answer.

A APPENDIX

B ADDITIONAL RELATED WORK

Mathematical reasoning (Chiang & Chen, 2018; Saxton et al., 2019) has been subject of interest for a long time. Faldu et al. (2021) summarizes the mathematical reasoning benchmarks that are in the form of math-word problems. In addition to this class of benchmarks, formal mathematics in the form of theorem-proofs (Rabe et al., 2020; Li et al., 2020; Polu & Sutskever, 2020; Welleck et al., 2021; Jiang et al., 2022; Wu et al., 2022) has been considered extensively. In this work we focus on algorithmic reasoning for arithmetic tasks and solving GSM8k (Cobbe et al., 2021b) problems.

Algorithmic reasoning is typically approached via using structured architectures such as graph neural networks(GNNs) and modifying the architecture align to the algorithms under consideration (Kaiser & Sutskever, 2015; Chiang & Chen, 2018; Xu et al., 2019; Gordon et al., 2019; Yan et al., 2020; Chen et al., 2020; Xhonneux et al., 2021; Veličković & Blundell, 2021) or to the input format (Thawani et al., 2021). However, in this work we focus on teaching algorithmic reasoning to general purpose transformer-based (Vaswani et al., 2017) models.

There has been some recent works investigating in-context learning phenomena. Razeghi et al. (2022) showed that the performance of LLMs on mathematical calculations correlates with term frequency in the training data. Min et al. (2022) investigate which parts of the (input, output) pairs in the prompt play a role in model's performance on 12 NLP tasks. Madaan & Yazdanbakhsh (2022) investigate this for chain-of-thought prompts and conclude that the combination of text and patterns together play a role. Jones & Steinhardt (2022) compares failure modes of LLMs to human biases in the context of few-shot prompts.

C ADDITION RESULTS ON TEACHING OTHER ALGORITHMS

This section includes additional details and figures for Section 3.2.

Multiplication: To choose a reasonable value of n for this experiment, we evaluate the model's zero-shot accuracy in $1 \times n$-digit multiplication (shown in Figure 8). We see that after $n = 3$, the zero-shot performance falls drastically. Thus, we restrict to $n \leq 3$ and restrict the problems we consider to have at least one of the two numbers in $a \times b$ be less than 1000. If one of the numbers has more than 3 digits, we break it down into groups of ≤ 3 digits and add the resulting sub-components appropriately. The prompt used for this experiment is displayed in Section G.7.

Parity: Similar to (Anil et al., 2022) we investigate the parity problem as an example of length generalization. We use algorithmic prompt for parity problem and compare its performance to that of scratchpad few-shot prompt as discussed in Anil et al. (2022)(Figure 9) as well as few-shot prompt-

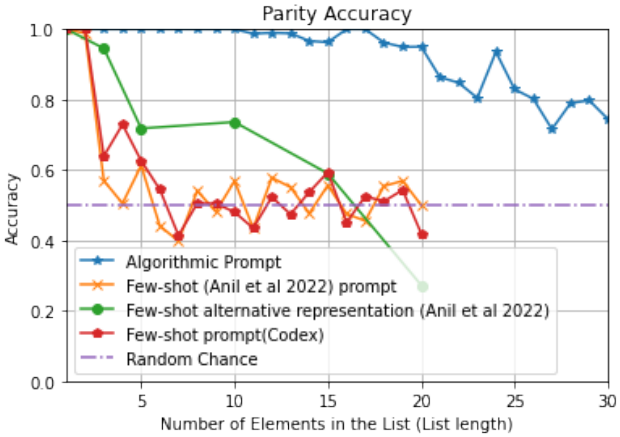


Figure 9: Investigating the performance of algorithmic prompt on parity problem and comparing it to scratchpad few-shot prompt of Anil et al. (2022) as well as few-shot prompting OpenAI's Codex. Each point on the plot corresponds to 100 random samples of a binary list of the same length. C.0.1, C.0.2 depict the prompts used for these three methods. The number of examples used in the prompt is two.

ing Codex. Figure 9 captures the performance of these three approaches on lists of varying sizes. The number of shots is two, each point represents average over 100 random samples and we use the same examples for all methods. We observe that algorithmic prompt significantly outperforms both baselines and while the baselines' performance reaches random chance (50%) around length 5, algorithmic prompt keeps accuracy of around 80% to lists of up to 30 digits. C.0.1, C.0.2 depict the prompts used for these three methods.

C.0.1 Algorithmic Prompt for Parity

```
Q: What is the parity on the list a=[1, 1, 0, 1, 0]?
A: We initialize s=
a=[1, 1, 0, 1, 0].  The first element of a is 1 so b=1.  s = s + b = 0 +
1 = 1.  s=1.
a=[1, 0, 1, 0].  The first element of a is 1 so b=1.  s = s + b = 1 + 1 =
0.  s=0.
a=[0, 1, 0].  The first element of a is 0 so b=0.  s = s + b = 0 + 0 = 0.
s=0.
a=[1, 0].  The first element of a is 1 so b=1.  s = s + b = 0 + 1 = 1.
s=1.
a=[0].  The first element of a is 0 so b=0.  s = s + b = 1 + 0 = 1.  s=1.
a=[] is empty.  Since the list a is empty and we have s=1, the parity is
1.
```

C.0.2 Scratchpad parity for Parity (Anil et al., 2022)

```
Q: What is the parity on the list a=[1, 1, 0, 1, 0]?
A: [1, 0, 0, 1, 1], the parity is 1.
Q: What is the parity on the list a=[0, 1, 1, 0, 0, 0, 0, 0]?
A: [0, 1, 0, 0, 0, 0, 0, 0] , the parity is 0.
```

D Additional results for Skill Accumulation

This section includes additional details and figures on skill accumulation from Section 4.

We study whether the superior performance of the algorithmic prompt can be attributed to the fact that it is much longer than the few-shot prompt. To control for this variation, we perform an ablation on the addition-subtraction of the few-shot baseline. We generate n examples of addition and subtraction, such that the total number of tokens is equal to the number of tokens used in the algorithmic prompt. We see the result in Figure 10, and find that having more few-shot examples does not improve performance.

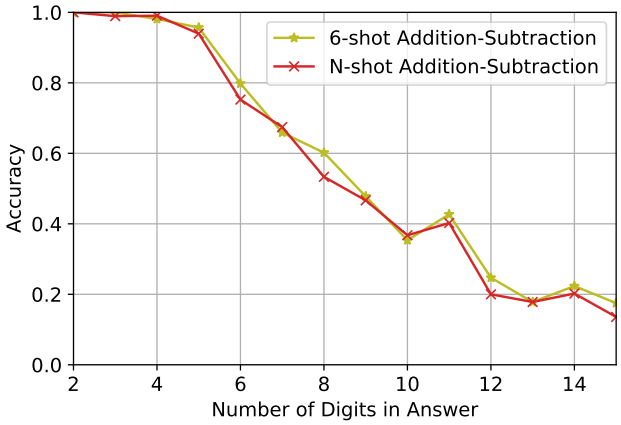


Figure 10: Accuracy on combined addition-subtraction questions using simple few-shot prompt. Since the algorithmic prompt uses more tokens in the prompt, we perform an ablation for the simple few-shot baseline and use n number of examples in the prompt, where n is chosen such that the prompt length matches the algorithmic prompt . We see that there is no improvements in performance beyond the 6 examples used in the baseline.

E Additional results for Skill Composition

This section includes additional details and figures on skill composition from Section 5. In Figure 11, we provide an illustrative demonstration of the change in the prompt when going from two-number addition to multi-number addition to multiplication-as-addition, showing the progression in complexity.

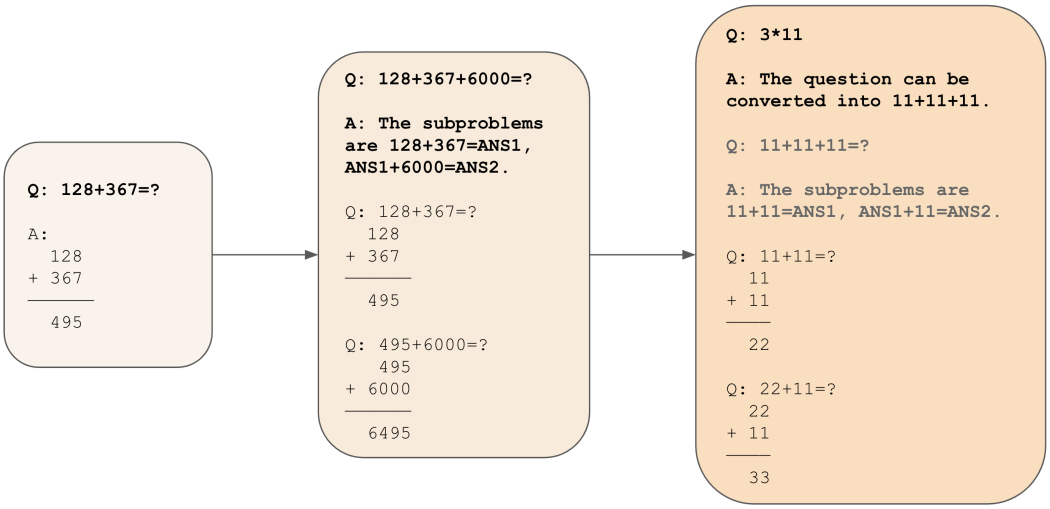
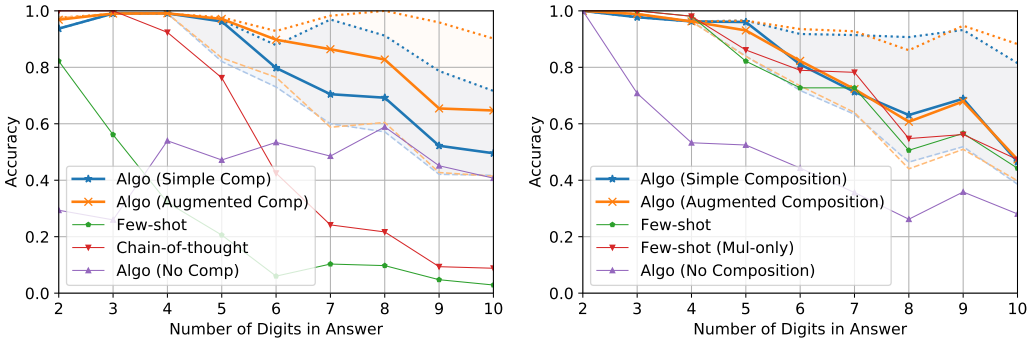


Figure 11: Illustration of the tasks and algorithmic prompting strategies considered for skill composition in Section 5

F Additional results for Tool Use

This section includes additional details and figures for tool use in Section 6.

In order to teach the model to use the additional algorithm for addition questions, we want to augment the chain-of-thought examples with algorithmic output for all addition equations. However, this would take up a lot of context without much gain in how well the addition algorithm is learned. Thus, we employ a strategy of choosing only 2 of the prompt examples to augment with algorithmic output, while another 6 examples are presented without algorithmic output. To differentiate the two types of approaches, we add the flag <ALGO> at the start of the answer for the 2 algorithmic output examples, and add the flag <NONALGO> for the others. At evaluation time, we evaluate performance



(a) Multi-number addition

(b) Multiplication-as-addition

Figure 12: Performance on compositions of skills. Due to length of the algorithmic output for this task, a number of the longest examples exceed the context length limit for Codex. We employ a second pass strategy to get a final answer for the incomplete questions, where we keep in-context only the last completed state from the first pass. The dotted lines consider only questions for which the model completes the output within one pass, and provide an upper bound on performance. The dashed lines consider all incomplete questions as false, and provide an lower bound on performance. We observe that although the algorithmic prompting methods are suffering from having to do a second pass for the longer samples in this task, they still demonstrate better generalization than the baseline methods. Note that for multiplication, we evaluate the algorithmic methods on a harder task than the few-shot baselines, since we force the model to convert the question into the addition of a number n times, while for other baselines we simply perform $1 \times n$-digit multiplication directly.

Q: Tommy has 3 toy cars. His neighbor, Jessie, has 3 cars too. Jessie's older brother has 5 more cars than Tommy and Jessie. How many cars do the three of them have altogether?
A: Tommy and Jessie have 3+3=6 cars. Jessie's brother has 5+6=11 cars. Altogether, they have 6+11=17 cars. The answer is 17.

Figure 13: An example question and answer pair from GSM8k with chain-of-thought rationale.

with algorithmic output by appending the <ALGO> flag to the end of the prompt, and we append <NONALGO> to get a non-algorithmic baseline using the same prompt. This flag-based strategy is simple yet effective, with 86% of <ALGO> examples and 0% of <NONALGO> examples exhibiting algorithmic output. See Section G.9 for the actual prompt.

G Prompt Examples

G.1 Chain-of-thought Prompt for Multi-add

Q: 9980+29=
A: Let's think step by step.
9980+29=10009
So, 9980+29=10009. The answer is 10009.
Q: 802+7145+6=
A: Let's think step by step.
802+7145=7947
7947+6=7953

```
Input:
128+367
Target:
<scratch>
1 2 8 + 3 6 7 , C: 0
1 2 + 3 6 , 5 C: 1
1 + 3 , 9 5 C: 0
, 4 9 5 C: 0
4 9 5
</scratch>4 9 5.
```

Scratchpad

```
Input:
128+367
Target:
<scratch>
[1,2,8] has 3 digits.
[3,6,7] has 3 digits.
[1,2,8] + [3,6,7] , C=0 , 8+7+0=15 , A->5 , C->1
[1,2] + [3,6] , A=[5] , C=1 , 2+6+1=9 , A->9 , C->0
[1] + [3] , A=[9,5] , C=0 , 1+3+0=4 , A->4 , C->0
[] + [] , A=[4,9,5] , C=0 , END
</scratch>
4 9 5
```

Detailed scratchpad w/ comma-delim

```
Problem: 128+367=?
Explanation:

Let's think step by
step.

128+367=128+300+67=428+6
7=495.

The final Answer is 495.
```

Chain-of-thought

```
Problem: 128+367=
...
Length of FN is 3. FN=[1,2,8]. Length of SN is 3.
SN=[3,6,7]. FN[3]=8. SN[3]=7. C[3]=0. Since 8+7+0=15,
15%10=5. Length of A is 1. Thus A=[5]. Since (15-5)/10=1,
C[2]=1.
Length of FN is 2. FN=[1,2]. Length of SN is 2. SN=[3,6].
FN[2]=2. SN[2]=6. C[2]=1. Since 2+6+1=9, 9%10=9. Length of
A is 2. Thus A=[9,5]. Since (9-9)/10=0, C[1]=0.
...
There are no more digits and C[0]=0. Thus the process is
complete. The final Answer is [4,9,5].
```

Algorithmic prompt

Figure 14: Examples of the two-number addition prompt using different techniques.

G.2 Algorithmic Prompt for Addition

```
Problem: 128+367=
Explanation:
The first number is 128, FN=[1,2,8]. The second number is 367,
SN=[3,6,7]. Since FN [1,2,8] has 3 digits, SN [3,6,7] has 3 digits, thus
the maximum number of digits is 3. In each subsequent step, we remove
one number from the end of FN and one from the end of SN.
Length of FN is 3. FN=[1,2,8]. Length of SN is 3. SN=[3,6,7].
FN[3]=8. SN[3]=7. C[3]=0. Since 8+7+0=15, 15>10, 15%10=5. Length of A
is 1. Thus A=[5]. Since (15-5)/10=1, C[2]=1.
Length of FN is 2. FN=[1,2]. Length of SN is 2. SN=[3,6]. FN[2]=2.
SN[2]=6. C[2]=1. Since 2+6+1=9, 9<10, 9%10=9. Length of A is 2. Thus
A=[9,5]. Since (9-9)/10=0, C[1]=0.
Length of FN is 1. FN=[1]. Length of SN is 1. SN=[3]. FN[1]=1.
SN[1]=3. C[1]=0. Since 1+3+0=4, 4<10, 4%10=4. Length of A is 3. Thus
A=[4,9,5]. Since (4-4)/10=0, C[0]=0.
There are no more digits and C[0]=0. Thus the process is complete.
Since there are no more operators, the problem is complete. The final
Answer is [4,9,5].
```

G.3 ALGORITHMIC PROMPT FOR ADDITION WITH IRREGULAR ERRORS

Problem: 128+367=
Explanation:
The first number is 128, FN=[1,2,8]. The second number is 367, SN=[3,6,7]. Since FN [1,2,8] has 3 digits, SN [3,6,7] has 3 digits, thus the maximum number of digits is 3. In each subsequent step, we remove one number from the end of FN and one from the end of SN.
Length of FN is 3. FN=[1,2,8]. Length of SN is 3. SN=[3,6,7]. FN[3]=8. SN[3]=7. C[3]=0. Since 8+6+0=15, 15>10, 15%10=5. Length of A is 1. Thus A=[5]. Since (15-5)/10=1, C[2]=1.
Length of FN is 2. FN=[1,2]. Length of SN is 2. SN=[3,6]. FN[2]=2. SN[2]=6. C[2]=1. Since 2+6+1=9, 9<10, 9%10=9. Length of A is 2. Thus A=[9,5]. Since (9-9)/10=0, C[1]=0.
Length of FN is 1. FN=[1]. Length of SN is 1. SN=[3]. FN[1]=1. SN[1]=3. C[1]=0. Since 1+2+0=4, 4<10, 4%10=4. Length of A is 3. Thus A=[4,9,5]. Since (4-4)/10=0, C[0]=0.
There are no more digits and C[0]=0. Thus the process is complete. Since there are no more operators, the problem is complete. The final Answer is [4,9,5].

G.4 ALGORITHMIC PROMPT FOR ADDITION WITH SYSTEMATIC ERRORS

Problem: 128+367=
Explanation:
The first number is 128, FN=[1,2,8]. The second number is 367, SN=[3,6,7]. Since FN [1,2,8] has 3 digits, SN [3,6,7] has 3 digits, thus the maximum number of digits is 3. In each subsequent step, we remove one number from the end of FN and one from the end of SN.
Length of FN is 3. FN=[1,2,8]. Length of SN is 3. SN=[3,6,7]. FN[3]=8. SN[3]=7. C[3]=0. Since 8+6+0=15, 15>10, 15%10=5. Length of A is 1. Thus A=[5]. Since (15-5)/10=1, C[2]=1.
Length of FN is 2. FN=[1,2]. Length of SN is 2. SN=[3,6]. FN[2]=2. SN[2]=6. C[2]=1. Since 2+5+1=9, 9<10, 9%10=9. Length of A is 2. Thus A=[9,5]. Since (9-9)/10=0, C[1]=0.
Length of FN is 1. FN=[1]. Length of SN is 1. SN=[3]. FN[1]=1. SN[1]=3. C[1]=0. Since 1+2+0=4, 4<10, 4%10=4. Length of A is 3. Thus A=[4,9,5]. Since (4-4)/10=0, C[0]=0.
There are no more digits and C[0]=0. Thus the process is complete. Since there are no more operators, the problem is complete. The final Answer is [4,9,5].

G.5 INSTRUCTION ADDITION PROMPT FOR ADDITION

```
The following are instructions for solving addition problems in the form
of x + y = z, where x, y, and z are positive integers.
We will use the standard algorithm for addition.  We align the numbers x
and y on the least significant digit, which is the ones digit.  Starting
from right to left, we go from the least significant digit to the most
significant digit and add the corresponding digits from each number.
When the sum of the two digits is greater than 9, a carry of 1 is
included in the sum of the next digits.  When there is only one digit
available from the two numbers, only that digit along with any carry
is included in the sum.  When all the digits are processed, only the
remaining carry if any shall be included in the sum.
For x + y = z where x = int(str(abc)), y = int(str(defg)), we can solve z
with the following steps:
1) c+g=w', w=w'%10
2) b+f+((w'-w)/10)=v', v=v'%10
3) a+e+((v'-v)/10)=u', u=u'%10
4) d+((u'-u)/10)=t', t=t'%10
5) s=(t'-t)/10
Thus, z = int(str(stuvw)).
The answer should be in the form below:
Q: What is abc+defg=?
A: abc
+defg
-------
stuvw
The answer is stuvw.
```

G.6 ALGORITHMIC PROMPT FOR ADDITION-SUBTRACTION

Problem: 9980+29=
Explanation:
The first number is 9980, adding commas between each number, FN=[9,9,8,0]. The second number is 29, adding commas between each number, SN=[2,9]. FN [9,9,8,0] has 4 digits, SN [2,9] has 2 digits, max is 4.
Len(FN)=4. FN=[9,9,8,0]. FN[4]=0. Len(SN)=2. SN=[2,9]. SN[2]=9. C[4]=0. Since 0+9+0=9, 9<10, 9%10=9. Len(A)=1. A=[9]. Since (9-9)/10=0, C[3]=0.
Len(FN)=3. FN=[9,9,8]. FN[3]=8. Len(SN)=1. SN=[2]. SN[1]=2. C[3]=0. Since 8+2+0=10, 10>10, 10%10=0. Len(A)=2. A=[0,9]. Since (10-0)/10=1, C[2]=1.
Len(FN)=2. FN=[9,9]. FN[2]=9. Len(SN)=0. SN=[]. SN[0]=empty. C[2]=1. Since 9+0+1=10, 10>10, 10%10=0. Len(A)=3. A=[0,0,9]. Since (10-0)/10=1, C[1]=1.
Len(FN)=1. FN=[9]. FN[1]=9. Len(SN)=0. SN=[]. SN[0]=empty. C[1]=1. Since 9+0+1=10, 10>10, 10%10=0. Len(A)=4. A=[0,0,0,9]. Since (10-0)/10=1, C[0]=1.
Len(FN)=0. FN=[]. FN[0]=empty. Len(SN)=0. SN=[]. SN[0]=empty. Since both FN and SN are empty, next. Since C[0]=1, continue. Len(A)=5. A=[1,0,0,0,9]. Since there are no - in A, we continue. The problem is complete. The final Answer is [1,0,0,0,9].
Problem: 29-570=
Explanation:
The first number is 29, adding commas between each number, FN=[2,9]. The second number is -570, adding commas between each number, SN=-[5,7,0]. FN [2,9] has 2 digits, SN -[5,7,0] has 3 digits, max is 3.
Len(FN)=2. FN=[2,9]. FN[2]=9. Len(SN)=3. SN=-[5,7,0]. SN[3]=-0. C[3]=0. Since 9-0+0=9, 9<10, 9%10=9. Len(A)=1. A=[9]. Since (9-9)/10=0, C[2]=0.
Len(FN)=1. FN=[2]. FN[1]=2. Len(SN)=2. SN=-[5,7]. SN[2]=-7. C[2]=0. Since 2-7+0=-5, -5<-10, -5%-10=-5. Len(A)=2. A=[-5,9]. Since (-5--5)/10=0, C[1]=0.
Len(FN)=0. FN=[]. FN[0]=empty. Len(SN)=1. SN=-[5]. SN[1]=-5. C[1]=0. Since 0-5+0=-5, -5<-10, -5%-10=-5. Len(A)=3. A=[-5,-5,9]. Since (-5--5)/10=0, C[0]=0.
Len(FN)=0. FN=[]. FN[0]=empty. Len(SN)=0. SN=-[]. SN[0]=empty. Since both FN and SN are empty, next. Since C[0]=0, the steps are done. Since there are - in A, we check the sign of the last step A[1]=-5. Since -5 is neg, we change the sign and process A from right to left. A=[-5,-5,9]=-[+5,+5,-9]. C[3]=0.
Len(A)=3. A=-[+5,+5,-9]. A[3]=-9. Since -9<0, B=10, C[2]=-1. Since C[3]=0, thus -9+10+0=1. Len(ANEW)=1. ANEW=-[1]. C[2]=-1.
Len(A)=2. A=-[+5,+5]. A[2]=+5. Since +5>0, B=0, C[1]=0. Since C[2]=-1, thus 5+0-1=4. Len(ANEW)=2. ANEW=-[4,1]. C[1]=0.
Len(A)=1. A=-[+5]. A[1]=+5. Since +5>0, B=0, C[0]=0. Since C[1]=0, thus 5+0+0=5. Len(ANEW)=3. ANEW=-[5,4,1]. C[0]=0.
Len(A)=0. A=-[]. Since A is empty, the problem is complete. The final Answer is -[5,4,1].

G.7 ALGORITHMIC PROMPT FOR MEMORIZED MULTIPLICATION

Q: 128*367=

```
Explanation:
FN=128, FN=[1,2,8].  SN=367, SN=[3,6,7].  Len(FN)=3, Len(SN)=3.  Max len
is 3.  Since 3=3, the lengths of two numbers are equal, we pick FN and
break [1,2,8] into 3//3=1 group of three and one group of 3%3=0 leftover
digits.  Since there are 0 leftover digits, from [1,2,8] we break the
first 0 digits as [][1,2,8], thus the leftover group is []=empty and the
main group is [1,2,8].  Since there is 3//3=1 group of three, we break
the main group into 1 group of 3 each:  [1,2,8].  Reformatting for each
main group, we have 128.  Thus, ignoring the empty group, the groups are
128.  The other number is the MULVAL, thus MULVAL=367.
The submulproblems are 128*367=MUL1.  There is 1 mul operator.
**START**
Submulproblem:  128*367=MUL1
FN=128, FN=[1,2,8].  Mulval=367.  Len(FN)=3.  P0=0.
Len(FN)=3.  FN=[1,2,8].  FN[3]=8.  8*367=2936.  P0=0, append 0 zero [] to
[2,9,3,6][]:  [2,9,3,6]=ADV1.
Len(FN)=2.  FN=[1,2].  FN[2]=2.  2*367=734.  P0=1, append 1 zero [0] to
[7,3,4|0]:  [7,3,4,0]=ADV2.
Len(FN)=1.  FN=[1].  FN[1]=1.  1*367=367.  P0=2, append 2 zero [0,0] to
[3,6,7|0,0]:  [3,6,7,0,0]=ADV3.
Len(FN)=0.  Done.
++START++
Addition Problem:  ADV1+ADV2+ADV3=
Explanation:
The subproblems are ADV1+ADV2=ANS1, ANS1+ADV3=ANS2.  There are 2 add
operators.
Subproblem:  ADV1+ADV2=ANS1
FN=ADV1, FN=[2,9,3,6].  SN=ADV2, SN=[7,3,4,0].  Len(FN)=4, Len(SN)=4, max
len is 4.
Len(FN)=4.  FN=[2,9,3,6].  Len(SN)=4.  SN=[7,3,4,0].  FN[4]=6.  SN[4]=0.
C[4]=0.  6+0+0=6, 6<10, 6%10=6.  Len(A)=1.  A=[6].  (6-6)/10=0, C[3]=0.
Len(FN)=3.  FN=[2,9,3].  Len(SN)=3.  SN=[7,3,4].  FN[3]=3.  SN[3]=4.
C[3]=0.  3+4+0=7, 7<10, 7%10=7.  Len(A)=2.  A=[7,6].  (7-7)/10=0, C[2]=0.
Len(FN)=2.  FN=[2,9].  Len(SN)=2.  SN=[7,3].  FN[2]=9.  SN[2]=3.  C[2]=0.
9+3+0=12, 12>10, 12%10=2.  Len(A)=3.  A=[2,7,6].  (12-2)/10=1, C[1]=1.
Len(FN)=1.  FN=[2].  Len(SN)=1.  SN=[7].  FN[1]=2.  SN[1]=7.  C[1]=1.
2+7+1=10, 10=10, 10%10=0.  Len(A)=4.  A=[0,2,7,6].  (10-0)/10=1, C[0]=1.
Len(FN)=0.  FN=[].  Len(SN)=0.  SN=[].  Both are empty.  C[0]=1.  Not
done.  Len(A)=5.  ANS1=[1,0,2,7,6].  Since there are 2 add operators and
we processed up to ANS1, continue.  The new FN is [1,0,2,7,6].
Subproblem:  ANS1+ADV3=ANS2
FN=ANS1, FN=[1,0,2,7,6].  SN=ADV3, SN=[3,6,7,0,0].  Len(FN)=5, Len(SN)=5,
max len is 5.  Len(FN)=5.  FN=[1,0,2,7,6].  Len(SN)=5.  SN=[3,6,7,0,0].
FN[5]=6.  SN[5]=0.  C[5]=0.  6+0+0=6, 6<10, 6%10=6.  Len(A)=1.  A=[6].
(6-6)/10=0, C[4]=0.
Len(FN)=4.  FN=[1,0,2,7].  Len(SN)=4.  SN=[3,6,7,0].  FN[4]=7.  SN[4]=0.
C[4]=0.  7+0+0=7, 7<10, 7%10=7.  Len(A)=2.  A=[7,6].  (7-7)/10=0, C[3]=0.
Len(FN)=3.  FN=[1,0,2].  Len(SN)=3.  SN=[3,6,7].  FN[3]=2.  SN[3]=7.
C[3]=0.  2+7+0=9, 9<10, 9%10=9.  Len(A)=3.  A=[9,7,6].  (9-9)/10=0,
C[2]=0.
Len(FN)=2.  FN=[1,0].  Len(SN)=2.  SN=[3,6].  FN[2]=0.  SN[2]=6.  C[2]=0.
0+6+0=6, 6<10, 6%10=6.  Len(A)=4.  A=[6,9,7,6].  (6-6)/10=0, C[1]=0.
Len(FN)=1.  FN=[1].  Len(SN)=1.  SN=[3].  FN[1]=1.  SN[1]=3.  C[1]=0.
1+3+0=4, 4<10, 4%10=4.  Len(A)=5.  A=[4,6,9,7,6].  (4-4)/10=0, C[0]=0.
Len(FN)=0.  FN=[].  Len(SN)=0.  SN=[].  Both are empty.  C[0]=0.  Done.
ANS2=[4,6,9,7,6].  Since there are add 2 operators and we processed up to
ANS2, complete.  The final ADDAnswer is [4,6,9,7,6].
++END++
**END**
```

continued on next page

```
MUL1=[4,6,9,7,6].  Since there is 1 mul operator and we processed up to
MUL1, complete.  We now combine the MUL results.  Since 1 mul operator,
we append 3*(1-1)=3*0=0 zeros to MUL1, MUL1=[4,6,9,7,6][]=[4,6,9,7,6].
Addition Mul Problem:  MUL1+EMPTY= Explanation:  The subproblems are
MUL1+EMPTY=ANS1.  There is 1 MA operator.  Since EMPTY is in the
equation, ANS1=MUL1=[4,6,9,7,6].  Since there is 1 MA operator and we
processed up to ANS1, complete.  The END Answer is [4,6,9,7,6].
```

G.8 ALGORITHMIC PROMPT PROMPT FOR MULTI-ADD AND MULTIPLY-AS-ADD

Problem: 128+367=

```
Explanation:
The first number is 128, FN=[1,2,8].  The second number is 367,
SN=[3,6,7].  Since FN [1,2,8] has 3 digits, SN [3,6,7] has 3 digits, thus
the maximum number of digits is 3.  In each subsequent step, we remove
one number from the end of FN and one from the end of SN.
Length of FN is 3.  FN=[1,2,8].  Length of SN is 3.  SN=[3,6,7].
FN[3]=8.  SN[3]=7.  C[3]=0.  Since 8+7+0=15, 15>10, 15Length of FN is
2.  FN=[1,2].  Length of SN is 2.  SN=[3,6].  FN[2]=2.  SN[2]=6.  C[2]=1.
Since 2+6+1=9, 9<10, 9Length of FN is 1.  FN=[1].  Length of SN is 1.
SN=[3].  FN[1]=1.  SN[1]=3.  C[1]=0.  Since 1+3+0=4, 4<10, 4There are no
more digits and C[0]=0.  Thus the process is complete.  The final Answer
is [4,9,5].
```

Problem: Problem: 9980+29=

```
Explanation:
The first number is 9980, FN=[9,9,8,0].  The second number is 29,
SN=[2,9].  Since FN [9,9,8,0] has 4 digits, SN [2,9] has 2 digits, thus
the maximum number of digits is 4.  In each subsequent step, we remove
one number from the end of FN and one from the end of SN.
Length of FN is 4.  FN=[9,9,8,0].  Length of SN is 2.  SN=[2,9].
FN[4]=0.  SN[4]=9.  C[4]=0.  Since 0+9+0=9, 9<10, 9Length of FN is 3.
FN=[9,9,8].  Length of SN is 1.  SN=[2].  FN[3]=8.  SN[3]=2.  C[3]=0.
Since 8+2+0=10, 10=10, 10Length of FN is 2.  FN=[9,9].  Length of SN
is 0.  SN=[].  FN[2]=9.  SN is empty.  C[2]=1.  Since 9+0+1=10, 10=10,
10Length of FN is 1.  FN=[9].  Length of SN is 0.  SN=[].  FN[1]=9.  SN
is empty.  C[1]=1.  Since 9+0+1=10, 10=10, 10There are no more digits,
but C[0]=1.  Length of A is 5.  Thus A=[1,0,0,0,9].  The final Answer is
[1,0,0,0,9].
```

Problem: 802+7145+6=

```
Explanation:
The subproblems are 802+7145=ANS1 and ANS1+6=ANS2.  There are 2
operators.
Subproblem:  802+7145=ANS1
The first number is 802, FN=[8,0,2].  The second number is 7145,
SN=[7,1,4,5].  Since FN=[8,0,2] has 3 digits, SN=[7,1,4,5] has 4 digits,
thus the maximum number of digits is 4.  In each subsequent step, we
remove one number from the end of FN and one from the end of SN.
```

continued on next page

Length of FN is 3. FN=[8,0,2]. Length of SN is 4. SN=[7,1,4,5].
FN[4]=2. SN[4]=5. C[4]=0. Since 2+5+0=7, 7<10, 7Length of FN is 2.
FN=[8,0]. Length of SN is 3. SN=[7,1,4]. FN[3]=0. SN[3]=4. C[3]=0.
Since 0+4+0=4, 4<10, 4Length of FN is 1. FN=[8]. Length of SN is 2.
SN=[7,1]. FN[2]=8. SN[2]=1. C[2]=0. Since 8+1+0=9, 9<10, 9Length of
FN is 0. FN=[]. Length of SN is 1. SN=[7]. FN is empty. SN[1]=7.
C[1]=0. Since 0+7+0=7, 7<10, 7There are no more digits and C[0]=0. Thus
the process is complete. Since there are 2 operators and we processed up
to ANS1, there are more operators to process. Thus, ANS1 is [7,9,4,7].
Subproblem: ANS1+6=ANS2
The first number is ANS1, FN=[7,9,4,7]. The second number is 6, SN=[6].
Since FN=[7,9,4,7] has 4 digits, SN=[6] has 1 digit, thus the maximum
number of digits is 4. In each subsequent step, we remove one number
from the end of FN and one from the end of SN.
Length of FN is 4. FN=[7,9,4,7]. Length of SN is 1. SN=[6].
FN[4]=7. SN[4]=6. C[4]=0. Since 7+6+0=13, 13>10, 13Length of FN is 3.
FN=[7,9,4]. Length of SN is 0. SN=[]. FN[3]=4. SN is empty. C[3]=1.
Since 4+0+1=5, 5<10, 5Length of FN is 2. FN=[7,9]. Length of SN is 0.
SN=[]. FN[2]=9. SN is empty. C[2]=0. Since 9+0+0=9, 9<10, 9Length
of FN is 1. FN=[7]. Length of SN is 0. SN=[]. FN[1]=7. SN is empty.
C[1]=0. Since 7+0+0=7, 7<10, 7There are no more digits and C[0]=0. Thus
the process is complete. Since there are 2 operators and we processed up
to ANS2, the problem is complete. The final Answer is [7,9,5,3].
Problem: 3*7=
Explanation:
The subproblems are 3*7=MS1. There is 1 * operator.
Subproblem: 3*7=MS1
Since the problem is multiplication, we find the smaller of the two
numbers and add the larger number as many times as the smaller number.
The first number is 3, FN=[3]=3. The second number is 7, SN=[7]=7.
Since 3 is smaller than 7, we rewrite the problem as 7 summed together
3 times: 7+7+7. We end at ANS(3-1)=2=ANS2.
The subproblems are 7+7=ANS1 and ANS1+7=ANS2. There are 2 operators.
Subproblem: 7+7=ANS1
The first number is 7, FN=[7]. The second number is 7, SN=[7]. Since
FN=[7] has 1 digit, SN=[7] has 1 digit, thus the maximum number of digits
is 1. In each subsequent step, we remove one number from the end of FN
and one from the end of SN.
Length of FN is 1. FN=[7]. Length of SN is 1. SN=[7]. FN[1]=7.
SN[1]=7. C[1]=0. Since 7+7+0=14, 14>10, 14There are no more digits and
C[0]=1. Length of A is 2. Thus A=[1,4].
There are no more digits and the process is complete. Since there are
2 operators and we processed up to ANS1, there are more operators to
process. Thus, ANS1 is [1,4].
Subproblem: ANS1+7=ANS2
The first number is ANS1, FN=[1,4]. The second number is 7, SN=[7].
Since FN=[1,4] has 2 digits, SN=[7] has 1 digit, thus the maximum number
of digits is 2. In each subsequent step, we remove one number from the
end of FN and one from the end of SN.
Length of FN is 2. FN=[1,4]. Length of SN is 1. SN=[7]. FN[2]=4.
SN[2]=7. C[2]=0. Since 4+7+0=11, 11>10, 11Length of FN is 1. FN=[1].
Length of SN is 0. SN=[]. FN[1]=1. SN is empty. C[1]=1. Since
1+0+1=2, 2<10, 2There are no more digits and C[0]=0. Thus the process
is complete. Since there are 2 operators and we processed up to ANS2,
the problem is complete. Since there is 1 * operator and we processed up
to MS1, the overall problem is complete. The final Answer is [2,1].

G.9 ALGORITHMIC PROMPT FOR GSM8K

```
Q: Tommy has 3 toy cars.  His neighbor, Jessie, has 3 cars too.  Jessie's
older brother has 5 more cars than Tommy and Jessie.  How many cars do
the three of them have altogether?
A: <NONALGO> Tommy and Jessie have 3+3=6 cars.  Jessie's brother has
5+6=11 cars.  Altogether, they have 6+11=17 cars.  The answer is 17.
Q: An electronic shop offers smartphones for $467 each, PCs are $128 more
expensive than smartphones, and advanced tablets are the prices of a
smartphone and a PC combined.  How much do you have to pay to buy one
of each of the three mentioned products?
A: <ALGO> To solve this problem, we need to find the prices of a PC and
an advanced tablet.  Then, we need to add the price of all three products
together.
The price of a PC is $128 more than a smartphone, thus the price of PC is
467+128.  We use the addition algorithm:
Problem:  467+128=
Explanation:
The subproblems are 467+128=ANS1.  There is 1 connecting operator.
Subproblem:  467+128=ANS1
The first number is 467, FN=[4,6,7].  The second number is 128,
SN=[1,2,8].  Since FN [3,6,7] has 3 digits, SN [1,2,8] has 3 digits, thus
the maximum number of digits is 3.  In each subsequent step, we remove
one number from the end of FN and one from the end of SN. Length of A is
0.
Length of FN is 3.  FN=[4,6,7].  FN[3]=7.  Length of SN is 3.
SN=[1,2,8].  SN[3]=8.  C[3]=0.  Since 7+8+0=15, 15>10, 15%10=5.  Length
of A is 1.  Thus A=[5].  Since (15-5)/10=1, C[2]=1.
Length of FN is 2.  FN=[4,6].  FN[2]=6.  Length of SN is 2.  SN=[1,2].
SN[2]=2.  C[2]=1.  Since 6+2+1=9, 9<10, 9%10=9.  Length of A is 2.  Thus
A=[9,5].  Since (9-9)/10=0, C[1]=0.
Length of FN is 1.  FN=[4].  FN[1]=4.  Length of SN is 1.  SN=[1].
SN[1]=1.  C[1]=0.  Since 4+1+0=5, 5<10, 5%10=5.  Length of A is 3.  Thus
A=[5,9,5].  Since (5-5)/10=0, C[0]=0.
There are no more digits and C[0]=0.  Thus the process is complete.
Since there is 1 operator and we processed up to ANS1, the problem is
complete.  The final Answer is [5,9,5].  Removing all 2 commas, we have
595.
The addition algorithm tells us that the price of a PC is 595.  Since
the price of an advanced tablet is the sum of a smartphone and a PC, its
price is 467+595.  We use the addition algorithm:  Problem:  467+595=
Explanation:
The subproblems are 467+595=ANS1.  There is 1 connecting operator.
Subproblem:  467+595=ANS1
The first number is 467, FN=[4,6,7].  The second number is 595,
SN=[5,9,5].  Since FN [4,6,7] has 3 digits, SN [5,9,5] has 3 digits, thus
the maximum number of digits is 3.  In each subsequent step, we remove
one number from the end of FN and one from the end of SN. Length of A is
0.
Length of FN is 3.  FN=[4,6,7].  FN[3]=7.  Length of SN is 3.
SN=[5,9,5].  SN[3]=5.  C[3]=0.  Since 7+5+0=12, 12>10, 12%10=2.  Length
of A is 1.  Thus A=[2].  Since (12-2)/10=1, C[2]=1.
Length of FN is 2.  FN=[4,6].  FN[2]=6.  Length of SN is 2.  SN=[5,9].
SN[2]=9.  C[2]=1.  Since 6+9+1=16, 16>10, 16%10=6.  Length of A is 2.
Thus A=[6,2].  Since (16-6)/10=1, C[1]=1.
Length of FN is 1.  FN=[4].  FN[1]=4.  Length of SN is 1.  SN=[5].
SN[1]=5.  C[1]=1.  Since 4+5+1=10, 10=10, 10%10=0.  Length of A is 3.
Thus A=[0,6,2].  Since (10-0)/10=1, C[0]=1.
There are no more digits, but C[0]=1.  Length of A is 4.  A=[1,0,6,2].
Thus the process is complete.  Since there is 1 operator and we processed
up to ANS1, the problem is complete.  The final Answer is [1,0,6,2].
Removing all 3 commas, we have 1062.
```
continued on next page

```
 The addition algorithm tells us that the price of an advanced tablet
is 1062.  To buy one of each of these products, you would have to pay
467+595+1062.  We use the addition algorithm:
Problem:  467+595+1062=
Explanation:
The subproblems are 467+595=ANS1, ANS1+1062=ANS2.  There are 2 connecting
operators.
Subproblem:  467+595=ANS1
The first number is 467, FN=[4,6,7].  The second number is 595,
SN=[5,9,5].  Since FN [4,6,7] has 3 digits, SN [5,9,5] has 3 digits, thus
the maximum number of digits is 3.  In each subsequent step, we remove
one number from the end of FN and one from the end of SN. Length of A is
0.
Length of FN is 3.  FN=[4,6,7].  FN[3]=7.  Length of SN is 3.
SN=[5,9,5].  SN[3]=5.  C[3]=0.  Since 7+5+0=12, 12>10, 12%10=2.  Length
of A is 1.  Thus A=[2].  Since (12-2)/10=1, C[2]=1.
Length of FN is 2.  FN=[4,6].  FN[2]=6.  Length of SN is 2.  SN=[5,9].
SN[2]=9.  C[2]=1.  Since 6+9+1=16, 16>10, 16%10=6.  Length of A is 2.
Thus A=[6,2].  Since (16-6)/10=1, C[1]=1.
Length of FN is 1.  FN=[4].  FN[1]=4.  Length of SN is 1.  SN=[5].
SN[1]=5.  C[1]=1.  Since 4+5+1=10, 10=10, 10%10=0.  Length of A is 3.
Thus A=[0,6,2].  Since (10-0)/10=1, C[0]=1.
There are no more digits, but C[0]=1.  Length of A is 4.  A=[1,0,6,2].
Thus the process is complete.  Since there are 2 operators and we
processed up to ANS1, there are more operators to process.  The new FN
is [1,0,6,2].
Subproblem:  ANS1+1062=ANS2
The first number is ANS1, FN=[1,0,6,2].  The second number is 1062,
SN=[1,0,6,2].  Since FN [1,0,6,2] has 4 digits, SN [1,0,6,2] has 4
digits, thus the maximum number of digits is 4.  In each subsequent
step, we remove one number from the end of FN and one from the end of
SN. Length of A is 0.
Length of FN is 4.  FN=[1,0,6,2].  FN[4]=2.  Length of SN is 4.
SN=[1,0,6,2].  SN[4]=2.  C[4]=0.
Since 2+2+0=4, 4<10, 4%10=4.  Length of A is 1.  Thus A=[4].  Since
(4-4)/10=0, C[3]=0.
Length of FN is 3.  FN=[1,0,6].  FN[3]=6.  Length of SN is 3.
SN=[1,0,6].  SN[3]=6.  C[3]=0.  Since 6+6+0=12, 12>10, 12%10=2.  Length
of A is 2.  Thus A=[2,4].  Since (12-2)/10=1, C[2]=1.
Length of FN is 2.  FN=[1,0].  FN[2]=0.  Length of SN is 2.  SN=[1,0].
SN[2]=0.  C[2]=1.  Since 0+0+1=1, 1<10, 1%10=1.  Length of A is 3.  Thus
A=[1,2,4].  Since (1-1)/10=0, C[1]=0.
Length of FN is 1.  FN=[1].  FN[1]=1.  Length of SN is 1.  SN=[1].
SN[1]=1.  C[1]=0.  Since 1+1+0=2, 2<10, 2%10=2.  Length of A is 4.  Thus
A=[2,1,2,4].  Since (2-2)/10=0, C[0]=0.
There are no more digits and C[0]=0.  Thus the process is complete.
Since there are 2 operators and we processed up to ANS2, the problem is
complete.  The final Answer is [2,1,2,4].
Removing all 3 commas, we have 2124.
The addition algorithm tells us that the sum of all the products is 2124.
The answer is 2124.
Q: Cally and Danny washed their clothes.  Cally has 10 white shirts, 5
colored shirts, 7 pairs of shorts, and 6 pairs of pants, while Danny has
6 white shirts, 8 colored shirts, 10 shorts, and 6 pairs of pants.  How
many clothes did they wash?
A: <NONALGO> They washed 10+6=16 white shirts.  They washed 5+8=13
colored shirts.  They washed 7+10=17 shorts.  They washed 6+6=12 pants.
Therefore, Cally and Danny washed a total of 16+13+17+12=58 clothes.  The
answer is 58.
Q: If there are four times as many red crayons as blue crayons in a box,
and there are 3 blue crayons.  How many crayons total are in the box?
A: <NONALGO> There are 4 times as many red crayons as blue crayons, which
means there are 3+3+3+3=12 red crayons.  Since there are 3 blue crayons
and 12 red crayons, in total there are 12+3=15 crayons.  The answer is
15.
```
continued on next page

```
 Q: Larry spent $5 for lunch and gave his brother $12.  How much did
Larry have at the beginning if he has $3 now?
A: <ALGO> To solve this problem, we need to find the total amount that
Larry spent and add it to the amount he has left.
Larry spent $5 for lunch and gave his brother $12, which means he spent
5+12.  We use the addition algorithm:
Problem:  5+12=
Explanation:
The subproblems are 5+12=ANS1.  There is 1 connecting operator.
Subproblem:  5+12=ANS1
The first number is 5, FN=[5].  The second number is 12, SN=[1,2].  Since
FN [5] has 1 digit, SN [1,2] has 2 digits, thus the maximum number of
digits is 2.  In each subsequent step, we remove one number from the end
of FN and one from the end of SN. Length of A is 0.
Length of FN is 1.  FN=[5].  FN[2]=5.  Length of SN is 2.  SN=[1,2].
SN[2]=2.  C[2]=0.  Since 5+2+0=7, 7<10, 7%10=7.  Length of A is 1.  Thus
A=[7].  Since (7-7)/10=0, C[1]=0.
Length of FN is 0.  FN=[].  FN[1]=0.  Length of SN is 1.  SN=[1].
SN[1]=1.  C[1]=0.  Since 0+1+0=1, 1<10, 1%10=1.  Length of A is 2.  Thus
A=[1,7].  Since (1-1)/10=0, C[0]=0.
There are no more digits and C[0]=0.  Thus the process is complete.
Since there is 1 operator and we processed up to ANS1, the problem is
complete.  The final Answer is [1,7].  Removing all 1 comma, we have 17.
The addition algorithm tells us that the amount spent is 17.  Larry has
$3 now, so he must have had 17+3 at the beginning.  We use the addition
algorithm:
Problem:  17+3=
Explanation:
The subproblems are 17+3=ANS1.  There is 1 connecting operator.
Subproblem:  17+3=ANS1
The first number is 17, FN=[1,7].  The second number is 3, SN=[3].  Since
FN [1,7] has 2 digits, SN [3] has 1 digit, thus the maximum number of
digits is 2.  In each subsequent step, we remove one number from the end
of FN and one from the end of SN. Length of A is 0.
Length of FN is 2.  FN=[1,7].  FN[2]=7.  Length of SN is 1.  SN=[3].
SN[2]=3.  C[2]=0.  Since 7+3+0=10, 10=10, 10%10=0.  Length of A is 1.
Thus A=[0].  Since (10-0)/10=1, C[1]=1.
Length of FN is 1.  FN=[1].  FN[1]=1.  Length of SN is 0.  SN=[].
SN[1]=0.  C[1]=1.  Since 1+0+1=2, 2<10, 2%10=2.  Length of A is 2.  Thus
A=[2,0].  Since (2-2)/10=0, C[0]=0.
There are no more digits and C[0]=0.  Thus the process is complete.
Since there is 1 operator and we processed up to ANS1, the problem is
complete.  The final Answer is [2,0].  Removing all 1 comma, we have 20.
The addition algorithm tells us that the total amount is 20.  The answer
is 20.
Q: Shawn has five toys.  For Christmas, he got two toys each from his mom
and dad.  How many toys does he have now?
A: <NONALGO> He has 5 toys.  He got 2 from mom, so after that he has
5+2=7 toys.  Then he got 2 more from dad, so in total he has 7+2=9 toys.
The answer is 9.
Q: Karen wanted to go out to get some fast food.  She pulls up to the
drive-through and orders a 5-dollar burger.  Her son then yelled out that
he wanted a 4-dollar sandwich, so it was added to the order.  Karen then
decided to order some drinks and opted for two 4-dollar smoothies.  What
is the total cost of Karen's fast-food order?
A: <NONALGO> Karen and her son order 5+4=9 dollars worth of food.  Karen
decides to buy 4+4=8 dollars worth of smoothies.  Thus, the total for
this order is 9+8=17 dollars.  The answer is 17.
Q: If there are 100 cars in the parking lot and 6 more cars arrive, how
many cars are in the parking lot?
A: <NONALGO> There are 100+6=106 cars in the parking lot.  The answer is
106.
```