Limitations of Language Models in Arithmetic and Symbolic Induction

Jing Qian*, **Hong Wang***, **Zekun Li, Shiyang Li, Xifeng Yan**
University of California, Santa Barbara
{jing_qian, hongwang600, zekunli, shiyangli, xyan}@cs.ucsb.edu

Abstract

Recent work has shown that large pretrained Language Models (LMs) can not only perform remarkably well on a range of Natural Language Processing (NLP) tasks but also start improving on reasoning tasks such as arithmetic induction, symbolic manipulation, and commonsense reasoning with increasing size of models (Wei et al., 2022; Chowdhery et al., 2022). However, it is still unclear what the underlying capabilities of these LMs are. Surprisingly, we find that these models have limitations on certain basic symbolic manipulation tasks such as copy, reverse, and addition. When the total number of symbols or repeating symbols increases, the model performance drops quickly. We investigate the potential causes behind this phenomenon and examine a set of possible methods, including explicit positional markers, fine-grained computation steps, and LMs with callable programs. Experimental results show that none of these techniques can solve the simplest addition induction problem completely. In the end, we introduce LMs with tutor, which demonstrates every single step of teaching. LMs with tutor is able to deliver 100% accuracy in situations of OOD and repeating symbols, shedding new insights on the boundary of large LMs in induction.

1 Introduction

Transformer-based large pretrained Language Models, such as GPT3 and T5 (Vaswani et al., 2017; Brown et al., 2020; Raffel et al., 2020), have been widely used as few-shot learners in many NLP tasks. Recent work even finds these models can achieve state-of-the-art performance in arithmetic and symbolic reasoning (Nye et al., 2021; Wei et al., 2022). Although these models exhibit surprisingly impressive capabilities in complex arithmetic reasoning tasks, such as MultiArith (Roy and Roth, 2015) and GSM8k (Cobbe et al., 2021), it has also been pointed out that they tend to make certain calculation errors and perform significantly worse when the number of math operations increases in equations (Wei et al., 2022). Brown et al. (2020) find that GPT3 displays strong proficiency in 2-digit arithmetic addition, but struggles in arithmetic addition on numbers with more than three digits. Nogueira et al. (2021) also observe that the fine-tuned T5 model can not correctly add or subtract arbitrarily long numbers. Larger models might perform better on the testing data, but worse on numbers that are longer than the training data (out-of-distribution, OOD) (Nogueira et al., 2021).

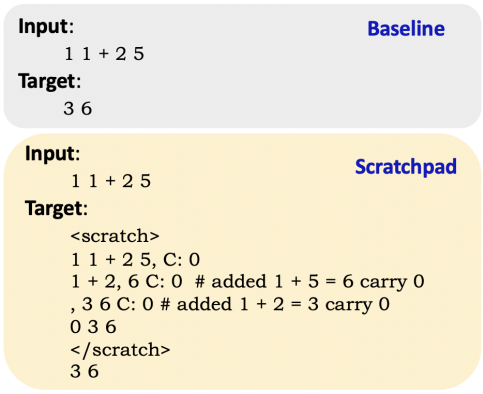

Figure 1: Examples of addition: the baseline setting (top) and Scratchpad (Nye et al., 2021) with intermediate steps (bottom). A similar method with more detailed demonstration is introduced in (Recchia, 2021).

Figure 1 shows two possible addition exemplars for LMs on addition problem. The scratchpad version gives more details on how humans do basic arithmetic. Nye et al. (2021) show that with more fine-grained demonstrations, the accuracy of addition can be improved dramatically with fine-tuning. Yet, it still can not achieve 100% on OOD data, even with thousands of training data points. Figure 2 shows the performance of GPT-3 and T5 on addition using the scratchpad version of training data. The problem becomes more severe when there are

* The first two authors (Jing and Hong) contributed equally to this work.

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 9285–9298

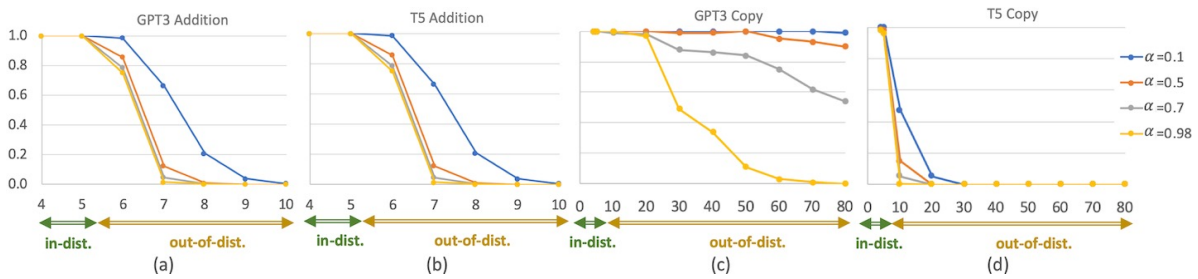

Figure 2: The horizontal axis is the number of digits and the vertical axis is the accuracy. The prompts for GPT3 consist of 4 examples. The T5 models are trained on 1-5 digits of up to 2,000 examples and each training example consists of random numbers in the format of 2 4 1. In-dist: in-distribution. Out-of-dist.: out-of-distribution (OOD). In-distribution refers to training on up to k-digit numbers and testing on up to k-digit numbers while out-of-distribution refers to training on up to k-digit numbers and testing on numbers with more digits. α indicates the repetition level of the examples. An example $x_1 \cdots x_n$ with n digits are sampled with the next digit probability $p(x_{i+1}|x_i) = \alpha$, when $x_{i+1} = x_i$; otherwise, $(1 - \alpha)/9$. Larger α indicates a higher repetition level.

repeating digits in the addition operands.

As the performance drops with repeating digits, we suspect that LMs might not handle the repeating symbols well. Figure 2 illustrates the performance of GPT-3 and T5 on the copy task, one of the simplest symbolic manipulation operations. GPT-3 and T5 still can not perform well on OOD. We further do a preliminary experiment where a T5 model is fine-tuned using the data containing repeating numbers of up to 80 digits, T5 still can not achieve 100% in-distribution accuracy on long repeating digits. The results indicate that there are two problems intervening: Transformers are not good at handling repeating symbols and OOD generalization. The repeating symbols can also be a problem even for in-distribution data. We believe that overcoming the aforementioned limitations is of critical importance for the future application of Transformer-based LMs to reasoning-intensive tasks such as data format conversion and robotic process automation.

In this paper, we investigate the potential causes behind this phenomenon and examine a set of possible mitigation solutions including fine-grained computation steps, positional markers, and LMs with callable programs. Since incorporating computation steps improves the OOD generalization in arithmetic addition (Nye et al., 2021), one possible direction is to provide more fine-grained computation steps in the fine-tuning data or the few-shot prompt. However, it may not be sufficient to alleviate the problem of repeating numbers. When a human does addition, the position of each digit is used to differentiate the repeating digits. However, the self-attention mechanism in the Transformer may not tell which "1" is referred to in the input. This prompts us to explore using positional markers to differentiate the important tokens. Using these two methods to augment the reasoning process, we find that the performance of pretrained LMs still can not reach satisfying results. Then we resort to a method where the copy operation is implemented as a primitive function and explore whether the LM can further boost its performance.

We experiment with three symbolic manipulation tasks: copying, reversing, and addition. Experimental results show that although generalization in these symbolic manipulation tasks is straightforward for humans, it is still challenging for LMs, and none of these mitigation methods fully solves the problems. In the end, we introduce LMs with tutor which demonstrates every single step of teaching, pinpointing where these digits come from. LMs with tutor is able to deliver 100% accuracy in situations of OOD and repeated symbols. In this design, LMs are used to generate actions that mimic operations in multiple tape Turing machines, rather than the intermediate results. These actions generate the intermediate results on tapes. We hope this could shed light on the capability of Transformer-based LMs in addition to providing large training datasets or scaling up the size of these models.

To conclude, our main contributions are:

- We identify a set of simple symbolic manipulation tasks and uncover the limitations of the LMs in arithmetic and symbolic induction.
- We examine a set of potential techniques including positional markers, fine-grained computation steps, and LMs with callable programs. Though they could mitigate the limitations of the LMs, none of them can completely

solve the generalization problem.

- Finally, we demonstrate that LMs with tutor is able to deliver 100% accuracy in situations of OOD and repeated symbols. Our analysis could inspire new thoughts to overcome the limitation of LMs in symbolic manipulation.

2 Related Work

Large Pretrained Language Models: Brown et al. (2020) show that GPT3 exhibits strong proficiency on 2-digit addition and subtraction using simply few-shot prompting, without any task-specific training. Furthermore, the larger the LM, the better the performance. Following GPT3, Chowdhery et al. (2022) further scale the Transformer-based LMs to a 540-billion parameter model, called Pathways Language Model (PaLM). Same as Brown et al. (2020), Chowdhery et al. (2022) find that scaling the LMs consistently results in better arithmetic reasoning ability with few-shot prompting. However, the reasoning ability of the large LMs is still limited. GPT3 struggles with 3-digit arithmetic and with direct prompting, even 540B PaLM can not achieve high performance on complex tasks requiring multi-step reasoning. Therefore Wei et al. (2022) propose the following prompting method for large pretrained LMs.

Chain-of-Thought Prompting: This prompting method provides a few chain-of-thought demonstrations, which is a series of intermediate reasoning steps, as exemplars in the prompting. Therefore, given a complex reasoning task, the model is allowed to calculate the intermediate results step-by-step before generating the final answer. With chain-of-thought prompting, a complex reasoning task is decomposed into a list of simple operations and LMs can derive these operations one by one. Kim et al. (2022) adopt faithful explanations that accurately represent the reasoning process behind solving a math word problem. Wei et al. (2022) show that combining chain-of-thought prompting and a sufficiently large LM, 540B PaLM, can significantly improve the LMs' reasoning ability on complex tasks, such as math word problems.

Fine-tuning with Large Training Datasets: Instead of few-shot prompting, another direction is to fine-tune large LMs with a sufficient amount of training data. Nogueira et al. (2021) fine-tune T5 with different ways of representing numbers, but even with the best-performing representation, the fine-tuned model can not achieve as good accuracy on out-of-distribution testing examples as in-distribution testing examples. Nye et al. (2021) propose to use Scratchpad to improve the out-of-distribution accuracy. Scratchpad combines step-by-step reasoning with fine-tuning. The training examples include the intermediate steps of an algorithm in target, so the model is trained to generate not only the final answer, but also the intermediate steps, which is similar to chain-of-thought, but requires more training data. Nye et al. (2021) show that using the training data augmented with intermediate steps significantly improves the model performance, but even with 100k augmented training examples for the addition task, the fine-tuned 1B LM still does not perform well on out-of-distribution addition. Our work is also related to Graves et al. (2014), which extends the capabilities of Recurrent Neural Networks to two simple symbolic manipulation tasks, copy and sort, by augmenting the model with external memory resources.

3 Mitigation Methods

3.1 Positional Markers

We first explore possible methods to mitigate the problem of repeating numbers. We introduce two types of positional markers: implicit positional markers and explicit ones.

Most Transformer-based LMs encode the positional information into positional vectors and add each of them to the corresponding word vector. Although large LMs have already incorporated positional encoding in the model architecture (Figure 3), results in Figure 2 indicate that the positional encoding commonly used in large LMs may not be sufficient to locate each repeating digit effectively. Instead of representing each token by the sum of its contextual token embedding and the position embedding, DeBERTa (He et al., 2021) represents each token with a token embedding and a position embedding, respectively, and the attention weights are computed using disentangled matrices based on both embeddings, respectively (Figure 3). In other words, the self-attention in DeBERTa is disentangled. With the disentangled relative position embeddings, the attention scores between tokens depend not only on the content but also on the relative position between the tokens, so the disentangled relative position embeddings act as implicit position markers within DeBERTa, which might make it easier for the model to learn the latent position relationship in the training data of the

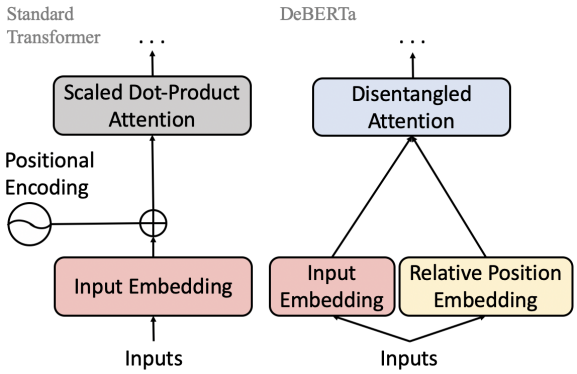

Figure 3: An illustration of standard Transformer attention (left) and DeBERTa disentangled attention (right).

symbolic manipulation tasks.

Although DeBERTa uses disentangled attention mechanism, it was not originally introduced to enhance the locating capability of LMs, so no pretraining task was specifically proposed for training the position embeddings in DeBERTa. This may potentially lead to its limited generalization ability on the induction tasks requiring accurate locating. Rather than relying on implicit positional markers, another, more straightforward approach is to add explicit positional markers in the model input. For example, the input string `2 2 2` is augmented with positional markers `A, B, C, ···`. We explore two methods of adding explicit positional markers:

Ordered marker: The markers are inserted into the input in order. `2 2 2` → `A 2 B 2 C 2`

Random marker: The markers are inserted into the input in random order. `2 2 2` → `E 2 X 2 J 2`

With the explicit positional markers, each repeating 2 becomes different for the model. When doing symbolic manipulation, the Transformer-based LMs can easily locate the digit by recognizing the explicit positional markers. Essentially, adding explicit positional markers breaks the repeating numbers into a non-repeating input sequence. This method is also related to pointer networks (Vinyals et al., 2015), which uses attention as a pointer to select the position indexes of the input tokens as the output. A hybrid pointer-generator network can also be leveraged to copy number from the source text, while retaining the ability to produce new numbers through the generator (See et al., 2017).

3.2 Fine-grained Computation Steps

We then explore possible methods to alleviate the OOD generalization problem. One observation is that the complexity of addition with long digits is larger than that of the 1-digit addition. Thus, the model should be given more computation time on the task when the numbers are large. The fine-tuned T5 and prompted GPT3 mentioned above, however, is required to generate the answer with a fixed amount of computation, so one possible direction to mitigate this limitation is to allow the model to operate step-by-step instead of generating the answer in one forward pass. For example, in k-digit addition, the model is allowed to break it down into k simple 1-digit addition and the model is allowed to generate k intermediate addition results to get the final answer.

Generating fine-grained computation steps can potentially alleviate the generalization problem, but may not contribute to the locating capability of the Transformer-based LMs. To mitigate the locating problem, we add positional markers to scratchpad (Nye et al., 2021) (Figure 4).

```
question: 1 1 + 2 5
solution:
convert 1 1 into ☞ 1, ☛ 1.
convert 2 5 into ☞ 2, ☛ 5.
☛ 1 5, carry 0, so 1 + 5 + 0 = 6. carry 0, step result 6.
combine 6 and result, get result 6.
☞ 1 2, carry 0, so 1 + 2 + 0 = 3. carry 0, step result 3.
combine 3 and result 6, get result 3 6.
carry 0, combine 0 and result 3 6, final result 3 6.
```

Figure 4: The prompt for GPT3 on the addition task. We use ☞ and ☛ to denote optional different markers as described in Section 3.1 if they are applied.

We also experiment a more comprehensive scheme where we directly copy the number associated with the explicit positional marker to its later appearance. For example, for the explicit marker S[B], we copy its value 1 to the later appearance in the fourth line as shown in Figure 5. More detail and experimental results are put in appendix A.4.

```
question: question: S[B] 1 S[A] 1 + T[B] 2 T[A] 5
solution:
S[A] 1 + T[A] 5 + Z[A] 0 = R[A] 6, Z[B] 0
S[B] 1 + T[B] 2 + Z[B] 0 = R[B] 3, Z[C] 0
result: Z[C] 0 R[B] 3 R[A] 6
```

Figure 5: The demonstration of comprehensive scheme for addition problem. Position markers are marked in red and reference markers are marked in green.

3.3 LM with Callable Programs

Since callable programs do not have the generalization problem, we combine LMs with callable programs to replace the basic symbolic operations when possible. For example, when combined with the fine-grained computation steps in the addition task, the convert, add, or combine operations can be considered callable programs. When the LM generates the text sequence `add(1,5)`, the callable function add will be invoked and return the result in text: `carry C: 0, result 6`.

Following the example in Section 3.2, with callable functions, the prompt format is as follows:

```
question: 1 1 + 2 5
solution:
call convert (1 1, 2 5), return ☞ (1 2), ☛ (1 5).
☛ (1 5), call add (1, 5), return carry C: 0, result 6.
call combine (6, ), return 6.
☞ (1 2), call add (C: 0, 1, 2), return carry C: 0, result 3.
call combine (3, 6), return 3 6.
call combine (C: 0, 3 6), return 3 6, final result 3 6.
```

Figure 6: The prompt for GPT3 on the addition task with callable programs. ☞ and ☛ are positional markers. Different callable programs (convert, add and combine) are marked in different colors, and the results they returned are underlined with the corresponding color.

Given a testing example, the prompted GPT3 first generates the solution step by step. During the process, the results of the function calls will be appended to the generated result to be used in the following steps. Callable programs can be viewed as decomposing a complex task to smaller, simpler jobs. The remaing issue is to learn chaining these smaller jobs together to complete the task.

Callable programs can guarantee the correctness of output given correct input for a given job. However, LMs may still suffer from the locating problem since the callable programs rely on LMs to decide which token to copy (Figure 11 in the appendix). Unfortunately, LMs cannot guarantee the correctness of this copy action.

3.4 LM with Tutor

Scratchpad (Nye et al., 2021) ignores the visual process when an elementary school tutor visually illustrates how to perform addition step by step: pinpointing where each digit in the output sequence comes from, adding single digits together and iterating. It turns out that these details and abstractions

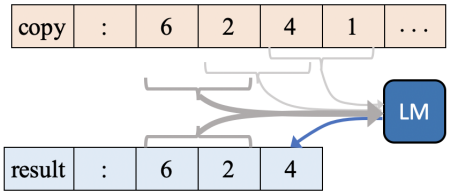

Figure 7: An illustration of doing copy with pattern matching.

are important in order to simplify the learning process and help kids learn addition in a few shots.

A tutor shows every single step visually and sometimes calls an already learned sub-module to complete a task. In this way, the hypothesis space between two consecutive steps can be dramatically simplified; hence the chance of learning a correct model can be improved.

Take copy as an example. Instead of providing a training example: `copy: 1 1 1 2 2 2 result: 1 1 1 2 2 2`, we need to demonstrate where the first 1, the second 1, and the third 1 in the output sequence come from, which exactly imitates the finest action a human could do to perform such an operation. Suppose there is a cursor placed at the beginning of the input sequence, a "rmov" operation moves the cursor one token to the right. A "cpy" operation copies a single digit to the output sequence. An "end" operation checks if the marker reaches the end of the sequence. "T" and "F" represent true and false respectively. We assume all these actions have been learned. Then a possible action sequence to complete the copy operation is as follows:

`rmov, end=F, cpy, rmov, end=F, cpy, . . . , rmov, end=T.`

This fine-grained action sequence accurately describes the whole copy operation. Certainly, there are other ways to perform copying. For example, instead of using a cursor, one can use a pattern match to perform the copy operation (Figure 7). We suspect that the copy operation learned from Transformer is following this pattern-matching approach, which is error-prone when the pattern has repeating symbols and when the long pattern is out-of-distribution. Positional markers do not help either as they seem unable to handle the OOD generalization problem.

If we take the action sequence "rmov, end=F, . . ." to train a Transformer for copying, the hypothesis space is simplified, thus making it possible to find the simplest model that can simulate the whole action sequence. This setting involves train-

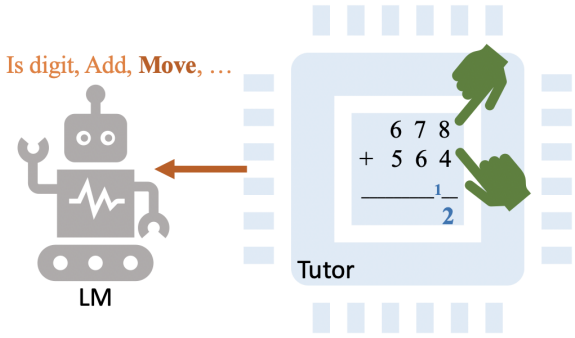

Figure 8: An illustration of the LM with Tutor method. With the tutor (right), the LM or just a transformer (left) generates an action sequence that simulates how humans do arithmetic addition.

ing a learner to predict the next action based on the input and the actions demonstrated by experts, which is similar to the setting of imitation learning (Pomerleau, 1988; Ross et al., 2011). Although there is no guarantee that Transformer can definitely find the correct model, the chance is much higher. One can also relate the setting with a multiple tape Turing machine where the state transition is conducted among the positions of tape heads and read/write operations. The Transformer is trained to learn such state transitions, thus completing the programming of a Turing machine.

As for the addition operation, a similar action sequence can be obtained to simulate how humans tutor kids do addition at an early age (Figure 8). Let "lmov" denote moving the cursor one token to the left. The "add" operation adds three single digits together, one from each of the two operands and the third one from the carry digit, appends the result to the output, and updates the carry digit. Assume "add" is a callable program as kids have learned how to do single digits addition. Suppose the cursor starts from the end of the operands. The entire action sequence looks like the following.
`lmov, end=F, add, lmov, end=F, add, . . . , lmov, end=T.`

The main difference between the tutor and the Scratchpad method (Nye et al., 2021) is the abstract callable function and detailed action sequence. The action sequence includes all the state transitions needed to complete the task. It perfectly overcomes the OOD issue and does not require many training examples in order to achieve 100% accuracy.

While there is a great effort to enlarge Transformer-based LMs such as PALM (Chowdhery et al., 2022) and Minerva (Lewkowycz et al., 2022), to improve the performance in symbolic and logical reasoning, our result reveals that it might be necessary to demonstrate the action sequence with reasonable abstraction to the Transformer to leverage its full strength.

In cases where action sequences are not available, e.g., only a problem specification is given, it might be more appropriate to develop an LLM (algorithm generator) to generate an algorithm sketch and then run another LLM to execute the sketch to get the answer. The sketch need not to be in the form of program codes. A human understandable step-by-step instruction is good enough. The sketch can be viewed as an intermediate model whose complexity is much smaller than the LLM itself. Hence it has a better chance of solving the generalization/OOD issue.

4 Experiments

In this section, we conduct experiments on three different problems including copying, addition, and another basic symbolic manipulation operation, reverse. We illustrate the limitation of LMs in symbolic and arithmetic induction and the improvement that could be achieved by the mitigation methods.

4.1 Copy Operation

Copying is the most basic operation. We experiment with the following methods and make sure each digit is tokenized into a single token by separating the digits with blanks:
GPT3: We prompt GPT3 to output the same tokens as the given input. Full prompt can be found in appendix (Figure 12).
DeBERTa / T5: The training example is as follows:
`copy: 1 2 3 4 result: 1 2 3 4`
T5 + ordered marker: The training data is augmented with explicit positional markers. `copy: A 1 B 2 C 3 result: A 1 B 2 C 3`
T5 + random marker: Same as above, but the augmented positional markers are in random order.
`copy: E 1 A 2 F 3 result: E 1 A 2 F 3`
T5 / GPT3 + tutor: The training and testing examples are as described in Section 3.4.

We experiment with the T5-base (220M) model, DeBERTa-base (140M) model, and GPT3 text-davinci-002. The models are initiated with the pretrained parameters and further fine-tuned on the training data. For GPT3 or T5 with tutor, the training data consists of 15 examples of up to 5 digits. For all the other T5 models and DeBERTa, the

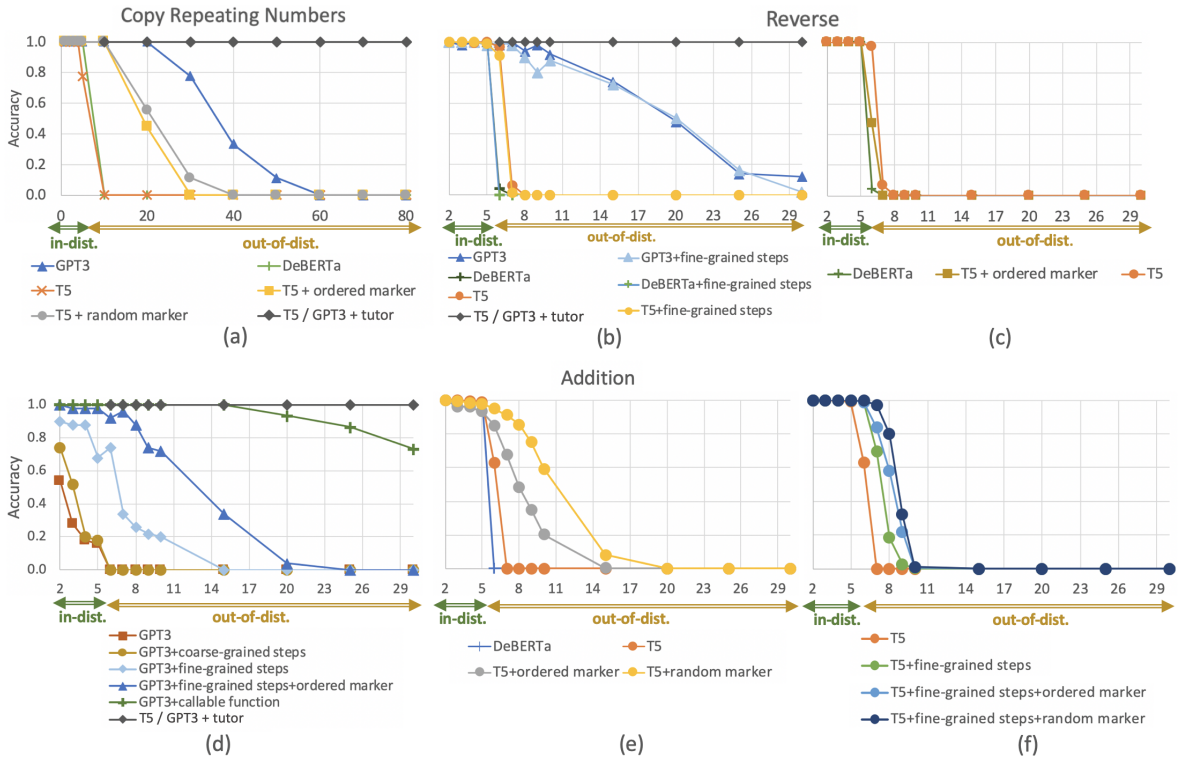

Figure 9: Experimental results. (a): results of copying repeating numbers. (b)(c): results of reversing the list. (d)(e)(f): results on arithmetic addition. The x-axis is the number of digits or number of items.

training data consists of 2,000 random numbers of up to 5 digits. We evaluate all the models on copying repeating numbers of up to 80 digits. The results are illustrated in Figure 9(a).

As shown in Figure 9(a), GPT3 achieves 100% accuracy on the in-distribution testing data (1-5 digits) but the fine-tuned T5 achieves 78% accuracy on the 5-digit repeating numbers although they are in-distribution. Augmented with random or ordered positional markers, the T5 models achieve 100% in-distribution accuracy, and so does using implicit positional markers (DeBERTa). This suggests that both implicit positional markers and explicit positional markers may help with the locating capability of LMs. However, using explicit positional markers, either ordered or random, the model exhibits significantly better generalization to OOD testing data whereas DeBERTa fails on OOD data. GPT3 exhibits better OOD generalization than T5 with positional markers but it does not generalize well beyond 30 digits. Both *T5 + tutor* and *GPT3 + tutor* keeps 100% accuracy on OOD testing data.

4.2 Addition

For arithmetic addition, we experiment with the following methods:

GPT3: We prompt GPT3 to directly output the sum for given addition equation. Full prompt can be found in appendix (Figure 13).

GPT3 + coarse-grained steps: The exemplar is similar to that in Figure 4, but the instructions for the result combination and the computation of the carry digit and step result are omitted.

GPT3 + fine-grained steps (+ ordered marker): The exemplar we use is as shown in Figure 4.

GPT3 + callable programs: The exemplar is shown in Figure 6.

DeBERTa / T5: The training data follows the format of the exemplar for GPT3.

DeBERTa / T5 + fine-grained steps: The training data used in this setting follow the format as the exemplar in *GPT3 + fine-grained steps*.

T5 + ordered / random marker: The training example is augmented with ordered or random markers. For example, `question: G 1 C 1 + G 2 C 5 result: G 3 C 6`. For the ordered marker, we apply it to the digits as the following: C 2 B 2 A 2.

T5 + fine-grained steps + ordered / random marker: The training data in this setting follow a similar format as the exemplar in *GPT3 + fine-grained steps + ordered marker*, but the positional markers can be in random order.

T5 / GPT3 + tutor: The training and testing examples are as described in Section 3.4.

The model settings are the same as in the above copy experiments. For LMs with tutor, the training data or prompt consists of 15 examples of up to 5 digits. In other settings, the training data consists of 1,000 examples of 1-5 digit addition and for GPT3, the prompt includes 4 examples. We evaluate all the models on the addition of up to 30 digits. The results are shown in Figure 9(d)(e)(f).

As shown in Figure 9(d), both coarse-grained and fine-grained computation steps contribute to the in-distribution performance of GPT3, and using finer-grained steps achieves larger performance gains on both in-distribution data and OOD data. The performance is further boosted with explicit positional markers. Experiments on T5 (Figure 9(e)(f)) also show the effectiveness of using explicit positional markers, with or without fine-grained computation steps, indicating that the explicit positional markers might make it easier for LMs to learn the induction in the arithmetic reasoning tasks. Similar to the results on the copying task, both DeBERTa and *DeBERTa + fine-grained steps* achieve near 100% in-distribution accuracy but 0% OOD accuracy, suggesting that the relative position embedding of DeBERTa might have limited OOD generalization ability. On T5, incorporating fine-grained computation steps does not improve the OOD performance as significantly as on GPT3 (Figure 9(f)). The reason might be that fine-tuning T5 tends to overfit more easily than prompting GPT3. Unsurprisingly, *GPT3 + callable programs* achieves much better OOD generalization. However, its OOD performance still degrades as the number of digits increases. Same as in the copy experiments, *LMs + tutor* keeps 100% accuracy on all the experimented numbers of digits.

4.3 Reverse List

Besides copying and addition, we also experiment with reversing. Reversing is similar to copying. Both require replicating the items in the input, but reversing might be more challenging than copying in the terms of locating. In copying, the distance between each source digit and the replicated digit is the same for each digit in the number. However, when reversing, the distance between the source item and the replicated item keeps increasing during the generation. For this problem, we experiment with the following methods:

GPT3: We prompt GPT3 to directly output the reversed list of items without intermediate steps. Full prompt can be found in appendix (Figure 14).

DeBERTa / T5: `reverse the list: bike, apple, book result: bike, cat, pen`

GPT3 / DeBERTa / T5 + fine-grained steps: The training example for T5 and the exemplar for GPT3 are shown in Figure 10.

```
reverse the list: bike, cat, pen
solution:
A is bike. B is cat. C is pen.
Now to reverse, change the order to:
C is pen. B is cat. A is bike.
Result: pen, cat, bike
```

Figure 10: The prompt for GPT3 on the reverse task with fine-grained steps.

T5 + ordered marker: The list items are augmented with the ordered positional markers in the input. `reverse the list: A bike, B cat, C pen result: pen, cat, bike`.

T5 / GPT3 + tutor: The training and testing examples are very similar to that for the copy task. The only difference is the direction for move operation. "rmov" in the copy task is replaced by "lmov" here.

The model settings are the same as in the above experiments and the training data consists of examples of 1-5 items, which are randomly sampled from a predefined list of single-token nouns. For LMs with tutor, the training data or prompt consists of 15 examples of up to 5 items. For T5, the training data consists of 1,000 examples. For GPT3, each prompt includes 4 examples. We evaluate all the models on reversing the list of up to 30 items. The results are shown in Figure 9(b)(c).

Although GPT3 can generalize to 80 digits on copying random numbers (Figure 2), it does not generalize well beyond 20 items on reversing, which suggests that reversing might require stronger locating capability than copying. This problem also occurs on DeBERTa and T5. When tested on the OOD data, the models tends to generate only a sublist of the input. Using fine-grained steps (Figure 9(b)) or positional markers, whether implicit or explicit (Figure 9(c)), does not significantly improve the generalization of the experimented models. The reason might be the increasing distance between the source item and the replicated item as stated above. Again, *LMs + tutor* maintains 100% accuracy throughout the experiments. We put more discussion about the results in appendix A.5 due to the page limit.

5 Conclusion

In this work, we explore the limitations of pretrained LMs on arithmetic reasoning and symbolic manipulation. We experiment with three simple symbolic manipulation tasks and show that improving the locating and induction capability of LMs can be important for further improving their performance. Our method that combines abstraction and finest-grained step-by-step tutoring demonstrates its potential to generalize correctly, shedding light on possible directions orthogonal to scaling up LMs for future work in this area.

6 Limitations

In this work, we experiment with GPT3, T5, and DeBERTa. Other large pretrained LMs, such as PaLM (Chowdhery et al., 2022), is not covered in this work. We do not experiment with methods such as fine-tuning GPT3 due to the computation cost. The main purpose of this work is to uncover and analyze the fundamental limitations of LMs on symbolic and arithmetic induction instead of improving their performance of reasoning tasks, so we do not directly compare the mitigation methods with the previous work such as scratchpad (Nye et al., 2021) and (Wei et al., 2022) in our experiments. We leave more advanced methods for future work.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways. *CoRR*, abs/2204.02311.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *CoRR*, abs/1410.5401.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. Deberta: decoding-enhanced bert with disentangled attention. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Bugeun Kim, Kyung Seo Ki, Sangkyu Rhim, and Gahgene Gweon. 2022. EPT-X: An expression-pointer transformer model that generates eXplanations for numbers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4442–4458.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay V. Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models. *CoRR*, abs/2206.14858.

Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. 2021. Investigating the limitations of the transformers with simple arithmetic tasks. *CoRR*, abs/2102.13019.

Maxwell I. Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. 2021. Show your work: Scratchpads for intermediate computation with language models. *CoRR*, abs/2112.00114.

Dean Pomerleau. 1988. ALVINN: an autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems 1, [NIPS Conference, Denver, Colorado, USA, 1988]*, pages 305–313. Morgan Kaufmann.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Gabriel Recchia. 2021. Teaching autoregressive language models complex tasks by demonstration. *Computing Research Repository*, abs/2109.02102. Version 3.

Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, volume 15 of *JMLR Proceedings*, pages 627–635. JMLR.org.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1743–1752. The Association for Computational Linguistics.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2692–2700.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

A Appendix

A.1 Error case for LM with callable program

Here we show one error case for LM with callable program in Figure 11.

```
question: . . . 6 1 8 . . . + . . . 6 1 9 . . .
prediction:
call convert ( . . . 6 1 8 . . . , . . . 6 1 9 . . . ) return . . .
. . .
☛ (8 9), call add (C:0, 8, 9), return carry C: 1, result 7.
call combine . . .
☞ (6 1), call add (C: 1, 6, 1), return carry C: 0, result 8.
# ERROR! Should be ☞ (1 1), call add (C:1, 1, 1) . . .
. . .
```

Figure 11: An error example of GPT3 with callable functions. The error is highlighted.

A.2 GPT3 prompts

Here we show the prompts of GPT3 used for copy, addition and reverse tasks in Figure 12, 13 and 14.

```
copy: 8 3 2 2
result: 8 3 2 2
copy: 7 7 7 7
result: 7 7 7 7
copy: 3 9 4 3 2
result: 3 9 4 3 2
copy: 6 6 6 6 6
result: 6 6 6 6 6
```

Figure 12: The prompt for GPT3 on the copy task.

```
question: 1 1 + 2 5
result: 3 6
question: 5 0 2 + 7 0 3
result: 1 2 0 5
question: 1 9 2 7 + 4 2 1 8
result: 6 1 4 5
question: 3 1 3 9 8 + 4 7 2 7 1
result: 7 8 6 6
```

Figure 13: The prompt for GPT3 on the addition task without intermediate steps.

```
reverse the list: bike, cat, pen
result: pen, cat, bike
reverse the list: chair, bike, apple, book
result: book, apple, bike, chair
reverse the list: book, phone, fish, orange, fish
result: fish, orange, fish, phone, book
```

Figure 14: The prompt for GPT3 on the reverse task without intermediate steps.

```
question: S[F] 5 S[E] 2 S[D] 8 S[C] 1 S[B] 7 S[A] 1 +
          T[F] 6 T[E] 5 T[D] 0 T[C] 2 T[B] 4 T[A] 5
solution:
S[A] 1 + T[A] 5 + Z[A] 0 = R[A] 6, Z[B] 0.
S[B] 7 + T[B] 4 + Z[B] 0 = R[B] 1, Z[C] 1.
S[C] 1 + T[C] 2 + Z[C] 1 = R[C] 4, Z[D] 0.
S[D] 8 + T[D] 0 + Z[D] 0 = R[D] 8, Z[E] 0.
S[E] 2 + T[E] 5 + Z[E] 0 = R[E] 7, Z[F] 0.
result: Z[F] 0 R[E] 7 R[D] 8 R[C] 4 R[B] 1 R[A] 6
```

Figure 15: Error case for T5 model with positional and reference marker on addition problem.

A.3 Experiment configuration

For fine-tuning the T5-base and DeBERTa model, we use the learning rate 5e-5, batch size 16, training epochs 200. The maximum generation length is set to 512. The checkpoints are evaluated every 1000 optimization steps. The random seed is fixed to 42. We use the implementation for HuggingFace (Wolf et al., 2020). For GPT3, we set temperature=0, top_p=1, frequency_penalty=0, and presence_penalty=0. All the experiments are conducted on NVIDIA RTX A6000 GPUs.

A.4 Reference marker

As shown in Figure 5, we apply two different markers in the demonstration. The positional marker is used to define the value stored in the marker, while reference marker is used to explicitly copy the value from the positional marker with the same name. Each number in this demonstration is uniquely marked with positional or reference marker. For the positional marker, the model needs to generate both the marker and its value. For the reference marker, the model only needs to generate the marker and the value will be explicitly copied from its corresponding positional marker.

Similar to previous experiments on the addition problem, we train the model on 1-5 digits and test its performance on both in-domain (1-5 digits) and out-of-domain (6-10 digits) settings. The experimental results show that the model is able to achieve 100% accuracy on in-domain data, but get 0% accuracy on out-of-domain data. We also tried to extend the in-domain to 10 digits and get the same results that the model can solve in-domain problems, but fail to generalize to out-of-domain.

We show one error case of this model in Figure 15, where the error step is highlighted in yellow. On this 6-digit addition problem, the model skipped the last digit and directly jump to the result, which causes the error. The problem is the model doesn't learn to how to generalize from 1-5 digits to 6 digits. Instead, it is overfitting to the training data, which makes it directly output the results after adding 5 digits. How to reduce the hypothesis space and force the model to learn to generalize to out-of-domain data would be one future research direction to solve this problem.

A.5 Discussion

From the experimental results, we observe that fine-grained computation steps may improve the LM's induction ability on the arithmetic reasoning tasks and the granularity of the steps has an impact on the performance improvement. Finer-grained computation steps may contribute to larger performance improvement.

Positional markers, whether implicit or explicit, improves LMs' in-distribution performance on all the symbolic manipulation tasks in our experiments. However, We find that augmented with the relative position embeddings, DeBERTa tends to face more severe over-fitting than T5 during fine-tuning. In the reversing experiment, using the T5 model without pretrained parameters, the fine-tuned model can not achieve a good in-distribution performance after 200k optimization steps. However, the DeBERTa model without pretrained parameters achieves 100% in-distribution accuracy within only 2k optimization steps while the OOD accuracy drops, indicating that it has overfitted within 2k optimization steps. In other words, the relative position embeddings in DeBERTa significantly improve the model's capacity of positions, which improves in-distribution performance on simple symbolic manipulation tasks, but may not generalize well on OOD data. Compared with the implicit positional markers (relative position embeddings in DeBERTa), explicit positional markers might have better OOD generalization ability. However, incorporating symbolic manipulation tasks in the LM pretraining stage might alleviate this problem, so incorporating implicit positional markers can still be a possible direction of improving the LM's performance on reasoning tasks requiring locating ability.

Using LM with callable programs exhibits strong OOD performance on addition, suggesting that the LMs' ability to perform simple symbolic operations, such as copying, splitting, and combining, can be critical for improving their performance on

reasoning tasks. How to further improve the LMs' performance on more complex reasoning tasks in this direction is left for future work.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
6

☒ A2. Did you discuss any potential risks of your work?
We don't think our work has any potential risks.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
1

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☒ Did you use or create scientific artifacts?

Left blank.

☐ B1. Did you cite the creators of artifacts you used?
No response.

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
No response.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
No response.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
No response.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
No response.

☐ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
No response.

C ☑ Did you run computational experiments?

4

☒ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Left blank.

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
A.3

☒ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
I reported the results from a single run

☒ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
No used.

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.