Improving Mathematics Tutoring With A Code Scratchpad

Shriyash Upadhyay
Martian
yash@withmartian.com

Etan Ginsberg
Martian
etan@withmartian.com

Chris Callison-Burch
University of Pennsylvania
ccb@upenn.edu

Abstract

Large language models can solve reasoning tasks (like math problems) more effectively when they are allowed to generate rationales. However, a good tutoring system should not just generate solutions, but should also generate explanations and should be able to correct and guide students. We show that providing a code scratchpad improves performance on each tutoring step with a gradeschool mathematics dataset. On these tutoring tasks, GPT-3 models provided with a code scratchpad significantly outperform those given only a language scratchpad (77.7% vs 48.7% cumulative accuracy).

1 Introduction

Intelligent Tutoring Systems (ITS) are known to be effective aids to learning, but are currently difficult and time consuming to create. Such systems can aid learning significantly despite limitations, improving student performance with a median improvement of 0.66 standard deviations (Kulik and Fletcher, 2016). However, many notable ITS (for example (Chaudhri et al., 2013)) have been limited due to the time-intensive and costly processes required to create them. Previous work on ITS has typically focused on rule-based methods. To the degree that large language models (LLMs) are used, it has been to generate additional rules for such systems. Recently, advances in natural language processing have pointed at the possibility of using LLMs as tutoring systems, most notably 1) the success of large language models in math world problem solving due to rationale generation (Rajani et al., 2019; Nye et al., 2021; Wei et al., 2022) and 2) the improved alignment of dialogue agents such as ChatGPT and Sparrow (Glaese et al., 2022). We conduct a feasibility study on the application of LLMs to tutoring in the context of mathematics at an elementary school level by investigating their performance on the tasks required by an ITS (see Figure 1).

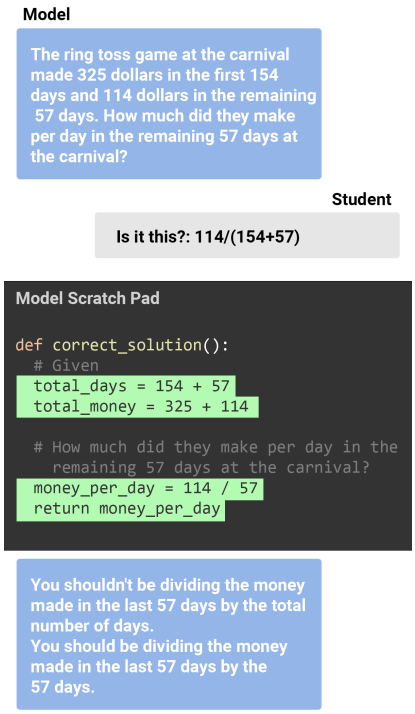

Figure 1: We evaluate the performance of two GPT-3 models on the sub-tasks present in an intelligent tutoring system, providing one with a text-only scratchpad and the other with a code scratchpad.

Our contributions are the following:

- We evaluate LLMs on the tasks present in an ITS by proving a mapping between the subtasks in an ITS and tasks which can be done by an LLM. Using this, we show that GPT-3 with a text-only scratchpad has a significant error rate when acting as a domain model and tutoring model.
- We show that using a code scratchpad instead of text-only ameliorates the errors in acting as a tutoring model. Combined with improved ability to solve math problems, this means GPT-3 makes a significantly better tutor with a code scratchpad (77.7% vs 48.7% cumulative accuracy on ITS sub-tasks).

Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023), pages 20–28

2 Related Work & Background

Early uses of NLP in ITS involved the use of knowledge-based and rule-based systems (Hartley and Sleeman, 1973). Such systems have shown to be pedagogically effective (Kulik and Fletcher, 2016), and as such they continue to constitute the majority of ITS today. Teaching and interacting with the student in an ITS takes place through some fixed set of interactions, often mediated by extracting keywords from user utterances or as goal-oriented dialogue systems. This tends to be the case in both knowledge-based ITS (Piramuthu, 2005; Chaudhri et al., 2013), and in rule-based systems (Jarvis et al., 2004; Stamper, 2006). For open-ended domains, Named Entity Recognition (NER) has been used to determine whether a student's open-ended response meets a set of constraints (Dzikovska et al., 2007). Techniques from NLP have also been used more selectively to implement features in these systems, such as machine translation for language learning (Moghrabi, 1998) and Automatic Speech Recognition (ASR) for audio-based tutors (Ward et al., 2011; Pradhan et al., 2016).

However, newer techniques such as LLMs have not found extensive use in implementing tutoring systems. This is despite the success of generative models such as GPT-3 (Brown et al., 2020) and PALM (Chowdhery et al., 2022) across a wide variety of tasks, the improvement in dialogue systems stemming from alignment as seen in models like ChatGPT and Sparrow (Glaese et al., 2022), and the success of LLMs (especially those that generate code) in the related domain of Math Word Problem Solving (Li et al., 2022; Gao et al., 2022). Much of the work on LLMs in education has focused on question generation as opposed to intelligent tutoring systems, for example (Dugan et al., 2022) for flashcard generation or (Sarsa et al., 2022) for programming exercises.

This may be the result of the difficulty in evaluating the quality of generations from LLMs, especially explanations for the answers that they give, as noted in (Lewkowycz et al., 2022). In this paper, we evaluate the ability of LLMs to serve as tutors, focusing on the evaluation of generated explanations and corrections.

3 Methodology

Intelligent Tutoring System. In order to evaluate the suitability of large language and code models to tutoring, we test how well those models do in the sub-tasks typically present in Intelligent Tutoring Systems.

Intelligent tutoring systems are typically composed of four components (Nkambou et al., 2010): the domain model, student model, tutoring model, and user interface model. The domain model consists of the actions and correct steps required to solve a problem. For example, in an ITS for mathematics the domain model might consist of all the relevant operations and the correction method of solving problems. The student model consists of the actions taken by the student (for example, the scratchpad the student is using to do their work). When the student deviates from the domain model, the tutoring model provides feedback (for example, telling a student what step they should take next or what a student did wrong in their scratchpad). Finally, the user interface model facilitates interaction between the user and the tutoring model (this might be the system which parses the scratchpad and then parlays feedback to the student).

We can instantiate a tutor using an LLM by creating each of the following parts. The user interface model is simply natural language. The domain model consists of problems with correct solutions (generated by the model), the student model consists of the language produced by the student, and the tutoring model consists of comparing domain and student models in text and producing feedback. We illustrate each of the parts of an ITS and how they can be performed by an LLM in Figure 4.

Dataset. Following previous work, we report our results on SVAMP (Patel et al., 2021). SVAMP is a challenge dataset consisting of 1000 math word problems designed to demonstrate the failures modes of word problem solving models. The dataset focuses on arithmetic word problems, i.e. those whose solutions are a combination of numerical values and the basic arithmetic operations ($+$, $-$, $\times$, $\div$). Examples of such problems can be found in Table 1. Each problem has both a body (containing the narrative that furnishes the relevant values and relationships) and the question being asked about that narrative. Each problem is also annotated with additional data, such as the correct numerical solution. The dataset also contains three types of "difficult" problems: problems with re-used values, problems with

Dave had 24 files and 13 apps on his phone. After deleting some apps and files he had 17 apps and 21 files left. How many files did he delete?
The grasshopper and the frog had a jumping contest. The grasshopper jumped 9 inches and the frog jumped 12 inches. How much farther did the frog jump than the grasshopper?
At the zoo, a cage had 95 snakes and 61 alligators. If 64 snakes were hiding How many snakes were not hiding?

Table 1: Examples of problems from the SVAMP dataset (Patel et al., 2021).

multiple operations, and problems with unused values.

Models. The large language model used in our experiments is GPT-3 (Brown et al., 2020). All experiments are run using the largest version of these models (the text scratchpad is generated with text-davinci-002 and the code scratchpad with code-davinci-002). For both models, decoding was done with nucleus sampling using p=1 (Holtzman et al., 2020). The temperature parameter was 0 and the frequency penalty was 0.5. The prompts used with each model can be found in Appendix A.

Scratchpads. Previous work has shown that providing models with a scratchpad where they can generate rationales for their answers improves their accuracy on reasoning tasks such as math word problem solving (Rajani et al., 2019; Nye et al., 2021; Wei et al., 2022). In our work, the scratchpads are a "thinking space" for models, which would not be shown to the students, but are used to compute answers or analyze student responses.

Scratchpads can take the form of text, code, or a combination of both. When the scratchpad is purely code, we extract an answer by running the code. When the scratchpad is text or a combination of both, the model produces an answer in the form of text.

Generating and Running Code. All code snippets generated in this paper's experiments are generated in the python programming language. If GPT-3 is used to generate runnable output, we generate GPT-3's response in a function named `solution`. Any code generated outside the `solution` function is not run. In order to prevent multiple solution functions from being generated, we stop generation whenever GPT-3 tries to open a multi-line comment using triple quotes (`"""`).

	Code	Text
Solved	79.4%	63.7%
Explained	98.9%	97.9%
Corrected	99.0%	78.1%
Cummulative	77.7%	48.7%

Table 2: Performance of GPT-3 with text/code scratchpads on each tutoring sub-task. The cummulative performance is the product of the performance on each sub-task.

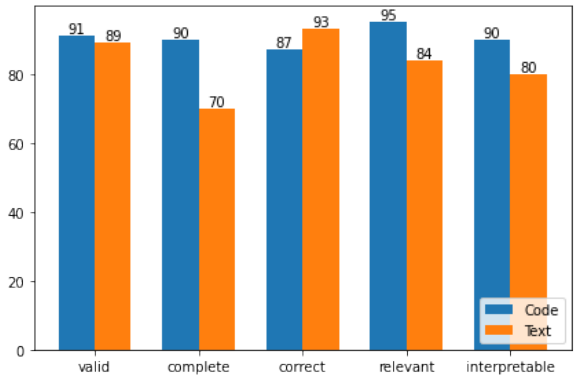

Figure 2: Results of our human evaluation for explanation generation. Numbers represent the percentage of annotations which provided a yes answer to each evaluation criterion.

4 Experiments

Our first experiment evaluates the difference in performance between text and code scratchpads in math problem solving. We evaluate, as is typical for math word problem solving, by measuring the percentage of numerically correct answers produced by the model. This is a necessary, but not sufficient, part of generating the domain model. The LLM should produce not only a correct answer, but should also provide a correct explanation to produce that answer. Therefore, our second experiment evaluates whether the model provides an acceptable explanation for its answer. Because we generate answers with GPT-3 by using CoT prompting, an explanation is automatically produced. For the code scratchpad, we generate an explanation by asking the model to convert the code used to produce an answer into plain English. These two experiments evaluate the ability of the LLMs to serve as a domain model.

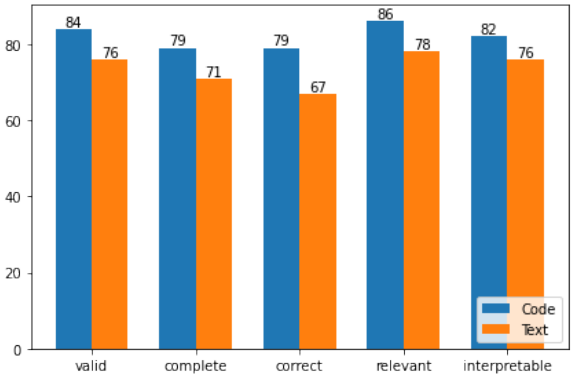

Figure 3: Results of our human evaluation for correction generation. Numbers represent the percentage of annotations which provided a yes answer to each evaluation criterion.

Our third experiment evaluates the ability of the LLMs to serve as tutoring models. We start with the correct answers and explanations provided by the model. For each question answered correctly, we prompt the models using poorly formed prompts in order to generate plausible incorrect answers (i.e. using the model to simulate the output of a student). Then, we provide the model with the incorrect answer and the correct answer, and prompt it to explain why the incorrect answer is wrong and to accordingly provide feedback to the student.

The first experiment is evaluated automatically, while the second and third experiments are evaluated by human annotators.

5 Evaluation

We tasked 208 annotators to evaluate the quality of explanations and corrections. Each annotator was shown 20 examples of explanations and later shown 20 examples of corrections. A total of 213 explanations and 190 corrections were evaluated in this way. We modify the question evaluation procedure in (Dugan et al., 2022) for evaluating explanations and asked the following yes/no questions:

1. (Valid) Does the explanation contain instructions which could be used to correctly answer the problem? It may also have other steps which are irrelevant or incorrect.
2. (Complete) Does the explanation explain all steps required to do the problem? That means the explanation is not missing any key steps a learner would need in order to solve such a problem.
3. (Correct) Does the explanation *not* contain any incorrect steps or incorrect explanation?
4. (Relevant) Does the explanation *not* contain information irrelevant to the problem.
5. (Interpretable) Would a student who is learning material at the level of this problem be able to understand the explanation?

If an annotator answered yes to all of the above questions, the explanation/correction was considered "acceptable"; otherwise, it was considered "unacceptable". Using Fleiss' κ, we observe moderate inter-annotator agreement ($\kappa = 0.21$).

In Table 2 we report the overall performance with each type of scratchpad on each sub-task. Code generation outperforms text generation on all sub-tasks.

In Figure 2 we report the detailed results of our evaluation for explanations. We can see that language and code scratchpads achieve similar performance in generating explanations. This is notable because of the difference in how the two models can create explanations. Text generation, by virtue of generating a Chain of Thought, comes with an explanation. Code generation requires an additional step of transforming code into text, which introduces an opportunity for more errors. This is reflected in the fact that explanations generated in text are more likely to be correct. However, code generation is much more likely to result in a complete explanation. This makes sense, as the model must explicitly list steps in code in order for the code to compile, while text is more prone to logical leaps or implicit steps.

In Figure 3 we report the detailed results of our evaluation for corrections. In contrast with explanation generation, when generating corrections, code scratchpads encounter fewer errors of all kinds than text ones.

6 Conclusion & Future Work

In this work we show that large language models can perform the tasks associated with traditional Intelligent Tutoring Systems (ITS). We show that models which use text scratchpads suffer from sbustantial errors in solving and correcting mathematical questions, and that these errors can be ameliorated through the use of code scratchpads. Nonetheless, code generation (while accurate enough to potentially useful as tool for authoring ITS) still suffers from significant errors.

Future work should seek to further explore the applicability of LLMs to tutoring. This includes developing both new evaluation methods and new methods of reducing errors.

7 Limitations

Testing Necessary, But Not Sufficient Conditions For Tutoring With LLMs. In this paper, we test the abilities of LLMs to perform the functions present in Intelligent tutoring systems, namely generating explanations and corrections. There are also other desirable properties, like the ability to answer direct questions from a student or the ability to present content engagingly, which are beyond the scope of this paper. Indeed, those properties are some of the areas where LLMs probably excel relative to traditional ITS. We have only explored a necessary condition – are models able to reliably teach – not a sufficient set of conditions for the evaluation of tutoring using an LLM.

Focusing On Mathematics. In this paper, we focus on tutoring in rudimentary mathematics. While this is useful – it is a necessary condition for a useful tutoring system, especially because arithmetic skills are used in almost all domains of learning – there are many other domains to which we might want to apply tutoring. LLMs may have greater or lesser aptitude in these domains than in arithmetic. Evaluation at the level of gradeschool mathematics tells us that these models are still error prone, but does not necessarily tell us how close they are to usefulness in tutoring other subjects (either more advanced mathematics or orthogonal subjects like history or writing).

Generalizing Text vs Code Results. We aim to examine the differences in ability of code scratchpads and text scratchpads for the purposes of tutoring. While this paper provides evidence in that direction, we only compare two GPT-3 models: text-davinci-002 and code-davinci-002. The amount of manual effort required to evaluate explanations and correction limited the number of comparisons we could conduct, as did the limited number of highly performant code/text generating models.

8 Ethics Statement

By offering a highly scalable and low-cost tutoring solution, ITS offer lower income and minority communities a critical resource in boosting educational outcomes that has historically only been available to wealthy students in the form of expensive individual private tutors. We hope that these advancements will reduce key educational disparities. It is also important in that vein to ensure that public schools with smaller budgets are given access to ITS systems in pilot trials. Instructors and students should become well-versed in using the technology in order to ensure successful expansion into such schools. Furthermore, advancements in model distillation and the creation of smaller language models will lead to lower costs for adoption for the schools that are most in need. Intelligent Tutoring Systems that run on generative AI models bring many of the same dangers of bias that are prevalent in models more generally. Gender and racial stereotypes can be invoked when students are presented with specific explanations. For example, a model may explain a math question that involved individuals choosing jobs through a hypothetical example that invokes a gender or racial stereotype based on the example given. However, recent advancements in alignment have made great strides in reducing this issue.

As these models become more widely available to students, there is an increased likelihood of students using these models for cheating on assignments that are supposed to be completed without outside resources. Unlike traditional plagiarism which can be checked by comparing document similarity, the use of generative AI to answer questions on exams and assignments is far more difficult to detect.

Lastly, discrepancies in model outputs and inaccurate answers given when some students use the ITS but not others can lead to misunderstandings and confusion amongst students. As a result, instructors should supervise the outputs given by the ITS to students. In the event that a student was supplied incorrect information by an ITS, that should be taken into account in grading that student's course material. Instructors should incorporate AI policies in their syllabi that outline acceptable uses of ITS systems, address the handling of potential inaccuracies from those systems, and ensure all students have access to the ITS systems.

By highlighting the limitations of large language models as tutoring systems, we hope our work will prevent the premature use of these technologies.

9 Acknowledgements

This research is based upon work supported in part by the the NSF (Award 1928631). Approved for Public Release, Distribution Unlimited. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the NSF or the U.S. Government.

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, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff 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. *ArXiv*, abs/2005.14165.

Vinay K. Chaudhri, Britte Haugan Cheng, Adam Overholtzer, Jeremy Roschelle, Aaron Spaulding, Peter Clark, Mark T. Greaves, and David Gunning. 2013. Inquire biology: A textbook that answers questions. *AI Mag.*, 34:55–72.

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 B Rao, Parker Barnes, Yi Tay, Noam M. Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Benton C. 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 García, 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 Oliveira Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Díaz, Orhan Firat, Michele Catasta, Jason Wei, Kathleen S. Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways. *ArXiv*, abs/2204.02311.

Liam Dugan, Eleni Miltsakaki, Shriyash Upadhyay, Etan Ginsberg, Hannah Gonzalez, DaHyeon Choi, Chuning Yuan, and Chris Callison-Burch. 2022. A feasibility study of answer-agnostic question generation for education. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1919–1926, Dublin, Ireland. Association for Computational Linguistics.

Myroslava O. Dzikovska, Charles B. Callaway, Elaine Farrow, Manuel Marques-Pita, Colin Matheson, and Johanna D. Moore. 2007. Adaptive tutorial dialogue systems using deep nlp techniques. In *North American Chapter of the Association for Computational Linguistics*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models. *ArXiv*, abs/2211.10435.

Amelia Glaese, Nat McAleese, Maja Trębacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Maribeth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, Lucy Campbell-Gillingham, Jonathan Uesato, Po-Sen Huang, Ramona Comanescu, Fan Yang, Abigail See, Sumanth Dathathri, Rory Greig, Charlie Chen, Doug Fritz, Jaume Sanchez Elias, Richard Green, Soňa Mokrá, Nicholas Fernando, Boxi Wu, Rachel Foley, Susannah Young, Iason Gabriel, William Isaac, John Mellor, Demis Hassabis, Koray Kavukcuoglu, Lisa Anne Hendricks, and Geoffrey Irving. 2022. Improving alignment of dialogue agents via targeted human judgements.

J. R. Hartley and Derek H. Sleeman. 1973. Towards more intelligent teaching systems. *International Journal of Human-computer Studies International Journal of Man-machine Studies*, 5:215–236.

Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. 2020. The curious case of neural text degeneration. *ArXiv*, abs/1904.09751.

Matthew P. Jarvis, Goss Nuzzo-Jones, and Neil T. Heffernan. 2004. Applying machine learning techniques to rule generation in intelligent tutoring systems. In *International Conference on Intelligent Tutoring Systems*.

James A. Kulik and John Dexter Fletcher. 2016. Effectiveness of intelligent tutoring systems. *Review of Educational Research*, 86:42 – 78.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Venkatesh 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. *ArXiv*, abs/2206.14858.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2022. On the advance of making language models better reasoners. *ArXiv*, abs/2206.02336.

Chadia Moghrabi. 1998. Using language resources in an intelligent tutoring system for french. In *ACL*.

Roger Nkambou, Jacqueline Bourdeau, and Riichiro Mizoguchi. 2010. *Advances in Intelligent Tutoring Systems*. Springer Berlin, Heidelberg.

Maxwell Nye, Anders 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. *ArXiv*, abs/2112.00114.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Selwyn Piramuthu. 2005. Knowledge-based web-enabled agents and intelligent tutoring systems. *IEEE Transactions on Education*, 48:750–756.

Sameer Pradhan, Ronald A. Cole, and Wayne H. Ward. 2016. My science tutor—learning science with a conversational virtual tutor. In *ACL*.

Nazneen Rajani, Bryan McCann, Caiming Xiong, and Richard Socher. 2019. Explain yourself! leveraging language models for commonsense reasoning. In *Annual Meeting of the Association for Computational Linguistics*.

Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic generation of programming exercises and code explanations using large language models. *Proceedings of the 2022 ACM Conference on International Computing Education Research - Volume 1*.

John C. Stamper. 2006. Automating the generation of production rules for intelligent tutoring systems.

Wayne H. Ward, Ronald A. Cole, Daniel Bolaños, Cindy Buchenroth-Martin, Edward Svirsky, Sarel van Vuuren, Timothy J. Weston, Jing Zheng, and Lee Becker. 2011. My science tutor: A conversational multimedia virtual tutor for elementary school science. *ACM Trans. Speech Lang. Process.*, 7:18:1–18:29.

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

A Prompts

A.1 Prompts Used For Math Problem Solving

Solving Math Problems With GPT-3

```
{problem.body} {problem.question}

A: Lets think step by step.
{model output}

So, the answer (in arabic numerals) is:
{model output}
```

Solving Math Problems With code

```
"""
{problem.body} {problem.question}
"""
{model output}

# So the answer (in arabic numerals) is: {model output}
```

A.2 Prompts Used For Explanation Generation

Converting Code Answers To English Explanations

```
"""
Write a function which computes and returns the solution to the following word problem:
At the zoo, a cage had 95 snakes and 61 alligators. If 64 snakes were hiding How many snakes were not hiding?
The function must return a single numerical value. It cannot print the answer.
"""
def solution():
  # Given
  snakes = 95
  alligators = 61
  hiding_snakes = 64


  # How many snakes were not hiding?
  return snakes - hiding_snakes

"""
Here's what the above code is doing:
1. The problem is asking how many snakes were not hiding. So, we need to find how many snakes were hiding and subtract it from how many snakes there were. (snakes - hiding_snakes)
2. The problem tells us that there were 95 snakes. (snakes = 95)
3. The problem tells us that 64 snakes were hiding. (hiding_snakes = 64)
4. So, the answer is 95 - 64 = 31.
"""

{answer}

"""
Here's what the above code is doing:
1. {model output}
```

A.3 Prompts Used To Generate Incorrect Answers

Generating example scratchpads using Code

```
"""
{problem.body} {problem.question}
"""
def solution():
    return {model output}
```

A.4 Prompts Used For Correction Generation

Correcting Solutions (used for both text and code)

```
{problem.body} {problem.question}
{correct_explanation}
{incorrect_answer}

What approach does the correct solution take:
{model outout}

What approach does the incorrect solution take:
{model output}

Why is the incorrect solution incorrect:
{model output}
```

B Annotation Interface

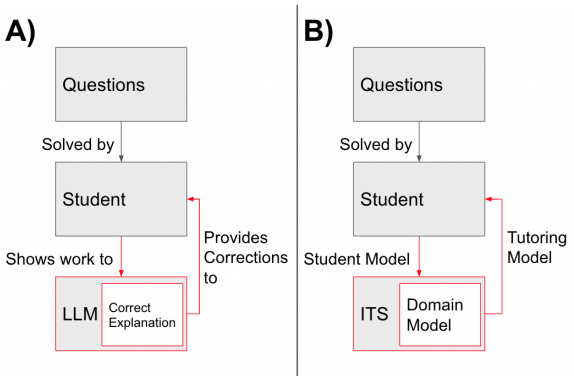

Figure 4: We evaluate the the performance of LLMs with text scratchpads and code scratchpads models in tutoring. (A) shows the parts of our system: given a question, a student produces an answer and the answer is shown to an LLM. The LLM first generates a solution to the question and a explanation for the solution. If the student gets the question wrong, the model also provides a correction. (B) shows how each of those steps corresponds to the parts of a traditional Intelligent Tutoring System (ITS). The red portion is what we evaluate: namely, the ability of the model to serve as a domain model (explanation) and tutoring model (correction).

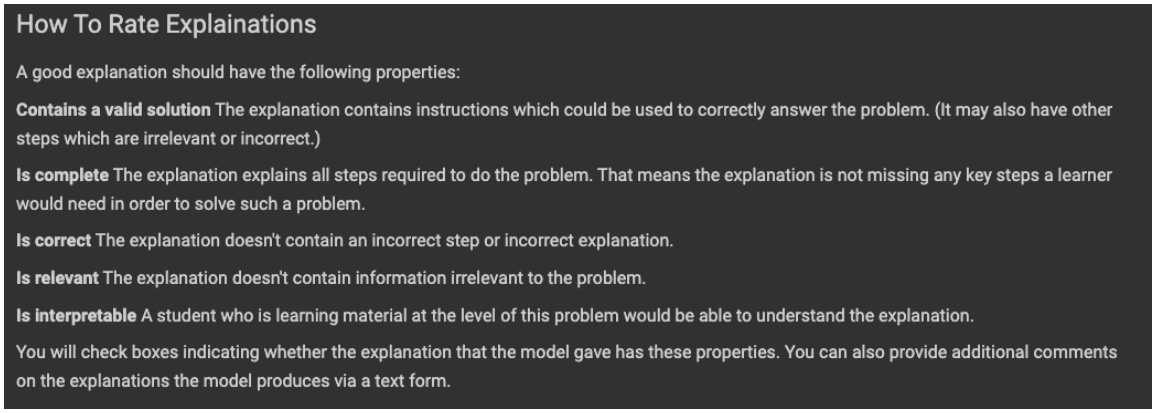

Figure 5: The annotation guidelines shown to annotators immediately before annotation. Complementary instructions were given prior to annotation.

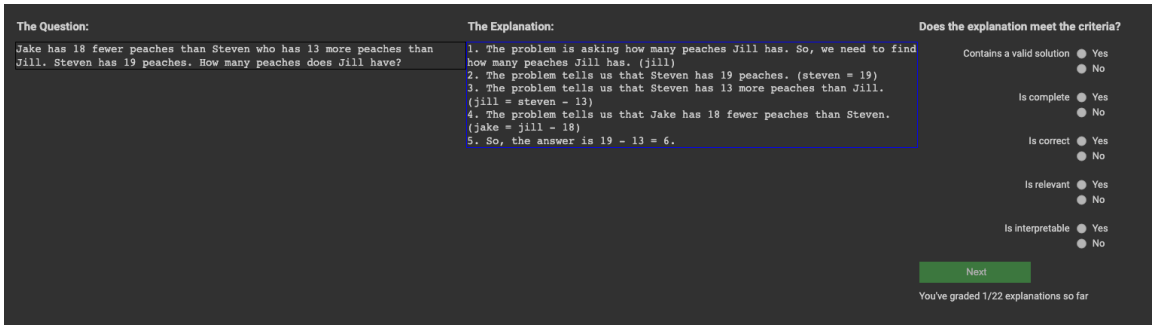

Figure 6: The annotation interface shown to annotators to annotate explanations.

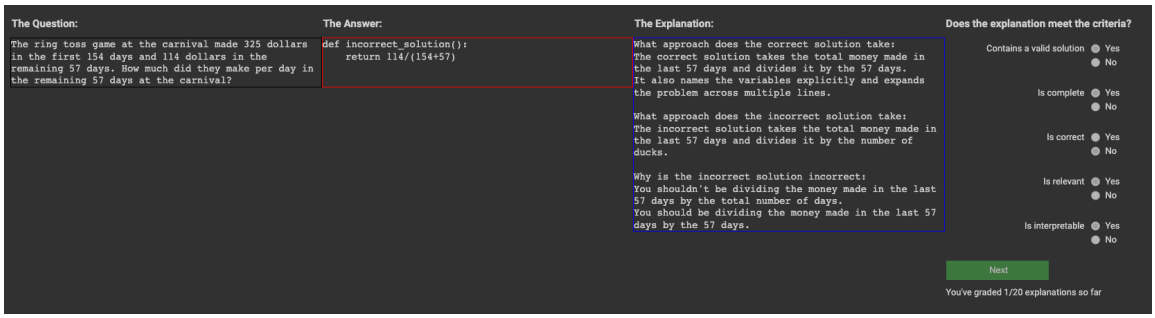

Figure 7: The annotation interface shown to annotators to annotate corrections.