An Automated Graphing System for Mathematical Pedagogy

Arya Bulusu
Merlyn Mind Inc.
arya.bulusu@merlyn.org

Brandon Man[*]
Massachusetts Institute of Technology
bm557@mit.edu

Ashish Jagmohan
Merlyn Mind Inc.
ashish@merlyn.org

Aditya Vempaty
Merlyn Mind Inc.
aditya@merlyn.org

Jennifer Mari-Wyka
Merlyn Mind Inc.
jennifer@merlyn.org

Abstract

Teachers use a variety of in-classroom technological tools in day-to-day instruction. The variety and complexity of operating these tools imposes a cognitive and time-overload, that teachers would rather spend with students. Pedagogical tool orchestration systems, based on generative AI, hold the promise of untethering teachers by enabling simple language-based operation of tools. Graphs are an essential tool in the classroom, allowing students to visualize and interact with mathematical concepts. In this paper, we present an automated graphing system for mathematical pedagogy. The system consists of an LLM and a mathematical solver used in conjunction with a math graphing tool to produce accurate visualizations from simple natural language commands. Our goal is to allow teachers to easily invoke math graphing tools through natural language, which is not possible through the use of a solver or an LLM alone. For benchmarking purposes, we create a dataset of graphing problems based on Common Core standards. We also develop an autoevaluator to easily evaluate the outputs of our system by comparing them to ground-truth expressions. Our results demonstrate the potential of tool usage with LLMs, as we show that incorporating a solver into the system results in significantly improved performance.

1 Introduction

Generative AI has significant potential to simplify the tools available in the classroom, allowing teachers to spend more time interacting with their students instead of their technology [2]. Our approach to incorporating generative AI into classroom technologies concerns the combination of large-language models (LLMs) and tool use. Recently, many frameworks have shown that connecting LLMs with external tools leads to more accurate, consistent responses and allows them to perform more difficult tasks [11], [10] [9], [13]. Through the use of tools, we can move towards an LLM system with the simplicity and reliability necessary for classroom use.

Teachers have many technological tools available to them in the classroom, but the extensive variety and complexity of these tools often makes it difficult to incorporate them into teaching. If tool-using is easier for teachers, they are untethered from their tools and more able to teach. Automated tool-using systems hold the promise of enabling language-controlled orchestration of pedagogical tools, reducing the overhead on teachers. In this paper, we present an automated graphing system for mathematical pedagogy. Graphs are an essential tool in the classroom, allowing students to visualize and interact

[*]Work was completed as an intern at Merlyn Mind Inc.

with mathematical concepts [4]. Our automated graphing system takes in utterances, converts them to mathematical expressions, and graphs them with the Desmos interface [3]. This simplifies the process of creating graphs in the classroom, allowing teachers to more easily incorporate math visualization techniques into their lessons without disrupting classroom flow.

Our contributions are:

- the design of a benchmark dataset of graphing problems based on Common Core standards [7]
- an automated graphing system combining an LLM with a mathematical solver
- an autoevaluation pipeline to evaluate different versions of our system
- results demonstrating the LLM+Solver system performance against an LLM-only system

Incorporating a solver into an LLM system provides a foundation of accuracy, as LLMs alone are incapable of reliably solving math problems. This allows the system to produce accurate graphs even for difficult, multi-step problems requiring complex reasoning. Mathematical solvers such as Wolfram Alpha [5] can provide accurate answers for many categories of problems, but they are not capable of understanding all types of natural language. As a result, an LLM is necessary to phrase queries to the solver based on the natural language instructions it is given. Our system also provides detailed explanations to accompany graphs, as it is vital for students to thoroughly understand the problem-solving process. By combining an LLM with an external tool, we produce an automated graphing system with strong potential for educational use.

2 Dataset

The Common Core standards [7] are a set of national educational standards describing what students are expected to know at each grade level, and they have been widely adopted in the United States. Based on the math Common Core standards, we identify a set of learning objectives that teachers use visualization tools to teach in the classroom. We use these categories as the basis of our dataset, creating approximately 10 questions per category to evaluate our system on.

Table 1: Question categories based on Common Core standards

	Standards
Construct Circles	7.G.B.4
Proportional Relationships	8.EE.B.5
Inequalities	7.EE.B.4
Lines	5.G.A.1, 5.G.A.2, 6.NS.C.8
Systems of Linear Equations	8.EE.C.8
Identify Zeros of Polynomials	HSA-APR.B.3, HSF-IF.C.7
Systems of Linear and Quadratic Equations	HSF-IF.C.7
Identify Intersections	HSA-REI.C.6, HSA-REI.C.7
Linear Inequality Systems	HSA-REI.D.12
Identify Extrema, Intercepts, and Asymptotes	HSF-IF.C.7
Transform Graphs	HSF-BF.B.3
Graph Tangents to Circles	HSG-C.A.4

The categories included in our datasets and the style of utterances were refined through teacher feedback. From these categories, we create two datasets; the first (referred to as the utterance-focused dataset) is focused on use cases a teacher might want to have available in the classroom. The utterances in this dataset are written as commands a teacher might say, as opposed to written-out problems for a student to solve. The dataset is mainly comprised of simpler, single-step problems that a teacher might use to demonstrate intermediate steps in the process of solving a problem.

Our other dataset (referred to as the textbook-focused dataset) is focused on multi-step, complicated problems that require tool use to solve. The main topics in this dataset are a superset of those in the teacher-focused dataset, but the problems are geared towards demonstrating the utility of tool use in

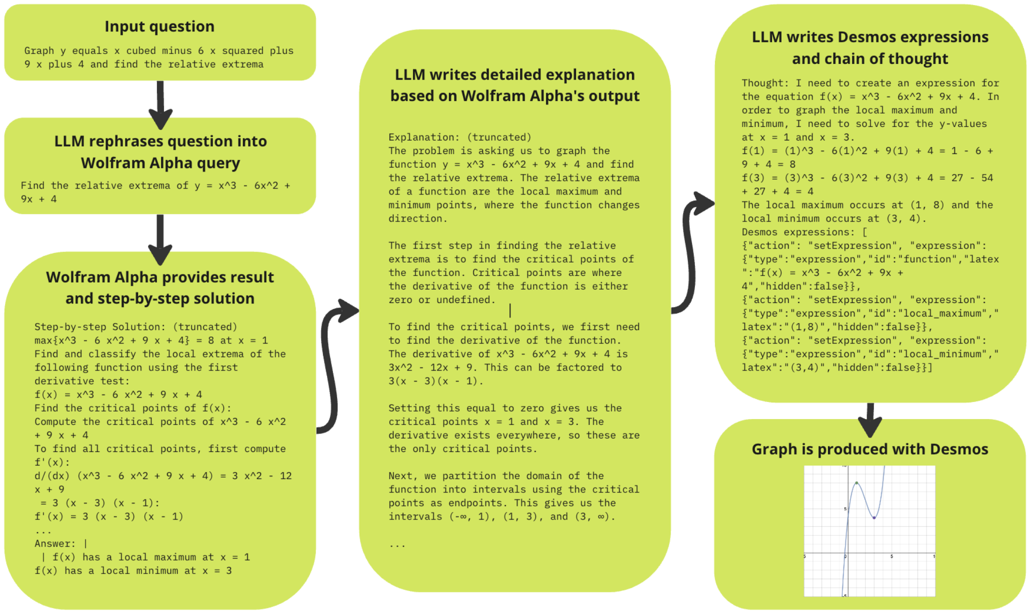
LLMs. In contrast to the utterance-focused dataset, we include word problems. The problems in this dataset are mainly taken from sources such as Khan Academy and IXL.

The datasets include a column for the problems and a column for the graph input associated with the problem. As the system is meant to be used through natural language commands, we also included a column with the utterance for the problem (e.g. "Graph $y = 5x^2 + 3$" vs. "Graph y equals five x squared plus three"). This column was automatically generated with GPT-4 [8] based on the original problem column and manually checked over for accuracy. The utterance-focused dataset contains 76 questions across seven categories, and the textbook-focused dataset contains 164 questions across fifteen categories.

Table 2: Example row from utterance-focused dataset

Processed Utterance	Natural Language Utterance	Graph Input
Reflect y = 5x - 4 across the y-axis	Reflect y equals five x minus four across the y-axis	y = -5x - 4

3 System



Input question
LLM rephrases question into Wolfram Alpha query
Wolfram Alpha provides result and step-by-step solution
LLM writes detailed explanation based on Wolfram Alpha's output
LLM writes Desmos expressions and chain of thought
Graph is produced with Desmos

```
Graph y equals x cubed minus 6 x squared plus
9 x plus 4 and find the relative extrema
```

```
Find the relative extrema of y = x^3 - 6x^2 +
9x + 4
```

```
Step-by-step Solution: (truncated)
max{x^3 - 6 x^2 + 9 x + 4} = 8 at x = 1
Find and classify the local extrema of the
following function using the first
derivative test:
f(x) = x^3 - 6 x^2 + 9 x + 4
Find the critical points of f(x):
Compute the critical points of x^3 - 6 x^2
+ 9 x + 4
To find all critical points, first compute
f'(x):
d/(dx) (x^3 - 6 x^2 + 9 x + 4) = 3 x^2 - 12
x + 9
 = 3 (x - 3) (x - 1):
f'(x) = 3 (x - 3) (x - 1)
...
Answer: |
 | f(x) has a local maximum at x = 1
f(x) has a local minimum at x = 3
```

```
Explanation: (truncated)
The problem is asking us to graph the
function y = x^3 - 6x^2 + 9x + 4 and find
the relative extrema. The relative extrema
of a function are the local maximum and
minimum points, where the function changes
direction.

The first step in finding the relative
extrema is to find the critical points of
the function. Critical points are where
the derivative of the function is either
zero or undefined.

To find the critical points, we first need
to find the derivative of the function.
The derivative of x^3 - 6x^2 + 9x + 4 is
3x^2 - 12x + 9. This can be factored to
3(x - 3)(x - 1).

Setting this equal to zero gives us the
critical points x = 1 and x = 3. The
derivative exists everywhere, so these are
the only critical points.

Next, we partition the domain of the
function into intervals using the critical
points as endpoints. This gives us the
intervals (-∞, 1), (1, 3), and (3, ∞).

...
```

```
Thought: I need to create an expression for
the equation f(x) = x^3 - 6x^2 + 9x + 4. In
order to graph the local maximum and
minimum, I need to solve for the y-values
at x = 1 and x = 3.
f(1) = (1)^3 - 6(1)^2 + 9(1) + 4 = 1 - 6 +
9 + 4 = 8
f(3) = (3)^3 - 6(3)^2 + 9(3) + 4 = 27 - 54
+ 27 + 4 = 4
The local maximum occurs at (1, 8) and the
local minimum occurs at (3, 4).
Desmos expressions: [
{"action": "setExpression", "expression":
{"type":"expression","id":"function","latex
":"f(x) = x^3 - 6x^2 + 9x +
4","hidden":false}},
{"action": "setExpression", "expression":
{"type":"expression","id":"local_maximum","
latex":"(1,8)","hidden":false}},
{"action": "setExpression", "expression":
{"type":"expression","id":"local_minimum","
latex":"(3,4)","hidden":false}}]
```

Figure 1: Overview of LLM+Solver automated graphing system

The system consists of three main components: the creation of the solver query, the generation of a written explanation based on the solver's output, and the generation of the Desmos graphing expressions based on the solver's output. In this paper, we use Wolfram Alpha as the solver and GPT-4 as the LLM, but the main principles of the system can be broadly applied to other tools and LLMs. We implemented this system in Python, using the OpenAI and Desmos APIs to create a problem-solving interface.

For a given problem, we create the solver query by prompting the LLM with instructions and a series of examples demonstrating how to write queries for certain math problems. These examples were chosen by identifying problems the LLM consistently misunderstood. The LLM is also provided with the spoken-utterance version of the problem and the calculator state. The calculator state contains the equations that have previously been graphed in the graphing window, and passing this state allows the system to incorporate this information into its problem-solving process. For the scope of this paper, we focus on single-turn utterances. Below is a truncated version of the prompt used to create the Wolfram Alpha query:

> Write a Wolfram Alpha query that can be used to solve the problem. The main purpose of the task is to find the numerical answer to the problem, not to graph the problem. When writing a query for a word problem, only include the necessary equation to solve the problem. Ensure that the query is acceptable by the Wolfram Alpha engine.
>
> For example, if you are asked:
>
> Graph $y = 6x^2 + 4$ and find the local maxima and minima.
>
> Calculator state: []
>
> You generate:
>
> Find the local maxima and minima of $y = 6x^2 + 4$

Once the query has been generated, we input it to our solver. Wolfram Alpha provides a set of pods for each query, with each pod containing a different category of information related to the query. Wolfram Alpha also provides step-by-step solutions for some problems. From these results, we extract the solution (generally the second pod, after the "Input Interpretation" pod) and the step-by-step solution if it is present.

To generate an explanation of the problem, we prompt the LLM with a zero-shot instruction. Along with the prompt, we provide the natural language utterance version of the problem, the calculator state, the numerical solution as given by the solver, and the solver's step-by-step solution, if it is present.

In the cases where Wolfram Alpha provides a step-by-step solution, the LLM only has to expand upon this solution by providing more detail and explaining the reasoning behind the steps. When there is no step-by-step solution given, it must write its own explanation from scratch based on the problem and numerical solution.

In order to generate the Desmos graphing expressions, we prompt the LLM with instructions and a set of examples. In the prompt, we ask for a chain-of-thought, which helps to generate more accurate expressions. Chain-of-thought prompting has been shown to improve the accuracy of LLMs' reasoning, especially with regards to math problems.[12][1] As with the explanation prompt, we also provide the natural language utterance version of the problem, the calculator state, the solver's numerical solution, and the solver's step-by-step solution, if there is one. The large number of examples in the prompt helps guide the LLM towards producing valid Desmos expressions. The provided examples were created by identifying common points of failure, and writing problems that demonstrate how to accurately deal with these issues.

4 Evaluation

4.1 Autoevaluation

Traditional evaluation metrics for text similarity fail for comparing mathematical statements due to the precise nature of math statements. Consider the statement 5=2+3. Lexical similarity metrics, such as Jaccard distance, would consider the statement 5=2+4 more similar than 5=4+1 since the former statements shares more words in common than the latter. Furthermore, many existing similarity metrics do not recognize common mathematical symbols as tokens and thus cannot be converted into a numerical representation. Similarly, directly examining the visual graph output through a multimodal approach is unlikely to be precise enough for our purposes. Although ChatGPT may be able to evaluate equivalence for simple expressions, its judgement becomes inconsistent for more complex expressions. As a result, evaluating the equations output by the automated graphing system at a large scale is nontrivial.

Due to these limiting factors, we create a new autoevaluation pipeline that can precisely compare two mathematical statements. We use the computer algebra system SymPy [6] to evaluate the mathematical equations output by the LLM in our LLM+Solver system when responding to given questions. In order to compare two equations, we use SymPy to isolate a variable and compare the resulting expressions on the other side of the equality.

Although this approach leads to accurate checking of math statements, an issue is that SymPy cannot parse poorly formatted equations, which the LLM in the LLM+Solver system may produce. To combat this, we use an LLM as a backup in the autoevaluation process, where if SymPy cannot parse an equation, it will let the LLM compare the two math statements and output a result.

We construct a set of ground truth evaluations by running all the questions in our datasets through the LLM-only system, and manually evaluating if the system's output matches the correct answer. This manually-benchmarked dataset allows us to run different versions of the autoevaluator on the dataset and check how its evaluations compare to our manually-written evaluations.

The simpler version of our autoevaluator only uses an LLM to compare two equations. Using GPT-4 as the LLM, we compare the results of LLM-only and LLM+SymPy autoevaluators on the entirety of our utterance-focused and textbook-focused datasets. In the table below, we display the dataset-wide results as well as the results for selected categories.

Table 3: Accuracy of LLM-only autoevaluator and LLM+SymPy autoevaluator as compared to manual evaluations

	Utterance-Focused Dataset	Textbook-Focused Dataset	Systems of Linear Equations	Graph Inverse Functions	Graph Lines
LLM-Only	77%	76%	50%	40%	**92%**
LLM+SymPy	**86%**	**88%**	**100%**	**80%**	85%

The addition of SymPy to the autoevaluation pipeline increases the accuracy of evaluations significantly on almost all categories, especially in Systems of Linear Equations. In general, the LLM+SymPy autoevaluator performs better than the LLM-only autoevaluator on problems that are well-structured but computationally difficult. These problems are easily interpretable and can be solved by SymPy, as it can easily handle complex algebraic manipulations. However, an LLM-only approach would likely make mistakes performing algebra.

We see a drop in performance for a few categories, such as Graph Lines. This decrease in accuracy is generally due to SymPy and the LLM both misunderstanding the formatting of the equation. An important point to note is that the LLM+SymPy autoevaluator almost never marks incorrect answers as correct. As a result, we can trust that nearly all the responses it marked as being correct are actually correct, and manually check if the responses it marked as being wrong are in fact wrong.

4.2 Results

In order to evaluate the LLM+Solver system, we run the autoevaluator and manually assess all of the outputs the autoevaluator marks as being incorrect. The autoevaluator greatly reduces the manual-assessment burden as we only have to look over a small subset of the total outputs, but this does introduce a bias as we do not double-check the entire dataset. However, false negatives (the autoevaluator determining a generated expression is equivalent to the ground-truth expression when they are actually inequivalent) are very rare as SymPy will only mark expressions equivalent if they are genuinely equivalent, and GPT-4 rarely marks inequivalent expressions as being equivalent. False positives are somewhat common, leading to the necessity of manual checking. Across both datasets, the autoevaluator marked 46% of problems as being incorrect. After manual evaluation, we found that the false positive rate was 48%. There were no false negatives.

We compare the performance of the LLM+Solver system to the results of the LLM-only system. The LLM-only system consists of directly prompting an LLM with instructions to write Desmos expressions and examples, while also providing the natural language utterance problem and the calculator state. This prompt is very similar to the prompt used in the LLM+Solver system, but with different examples as both systems have differing inputs. Although the framework of the system can be applied to LLMs and solvers broadly, in this paper we evaluate using GPT-4 as the LLM and Wolfram Alpha as the solver. Below are the results for the LLM-only system and the LLM+Solver systems:

Table 4: Accuracy of LLM-only and LLM+Solver models

	LLM-only	LLM+Solver
Utterance-Focused Dataset	63%	**85%**
Textbook-Focused Dataset	55%	**75%**

Table 5: Accuracy of individual categories in utterance-focused dataset

	LLM-only	LLM+Solver
Graph Circles	92%	**100%**
Transform Shapes	64%	**73%**
Intersections of Lines	20%	**100%**
Graph Lines	92%	**100%**
Local Minima and Maxima	60%	**90%**
X Intercepts, Y Intercepts	50%	**80%**

Table 6: Accuracy of individual categories in textbook-focused dataset

	LLM-only	LLM+Solver
Proportional Relationships	100%	100%
Linear Inequality Systems	**50%**	0%
Graph Inequalities	**70%**	50%
Graph Lines	84%	**93%**
Graph Polynomials + Identify Zeros	50%	**80%**
Systems of Linear + Quadratic Equations	0%	**85%**
Graph Circles	100%	100%
Transformations of Functions	91%	91%
Graph Inverse Functions	100%	100%
Tangents to Parabolas	0%	0%
Tangents to Circles	15%	**77%**
Local Minima and Maxima	10%	**100%**
Linear and Nonlinear Functions	50%	**60%**
Systems of Linear Equations	20%	**90%**
Rigid Transformations + Dilations	50%	**90%**

The addition of the solver results in a significant performance increase on both the utterance-focused and textbook-focused datasets. The greatest performance increase occurs in categories such as Local Minima and Maxima, X and Y intercepts, Intersections of Lines, and Tangents to Circles. These problems require complex calculations which are difficult for GPT-4 to carry out by itself, meaning that GPT-4 will frequently get them wrong. However, it is very easy to write a Wolfram Alpha query to solve these problems. As a result, the LLM+Solver system shows strong improvement over the LLM-only system for these categories.

The LLM-only system struggles the most in problems that require complicated reasoning and calculations to solve, such as the categories Tangents to Parabolas and Systems of Linear + Quadratic Equations. It tends to fail either by solving the problem with an incorrect method or executing calculations incorrectly. It succeeds in categories that require minimal reasoning and mathematical calculations, such as Proportional Relationships and Graph Inverse Functions.

For many categories, the performance of the LLM-only system was strong to begin with, such as Circles, Proportional Relationships, and Graph Lines. These categories contain simple problems with little mathematical calculation, so GPT-4 is able to succeed at these problems without the help of a solver. There are some categories for which there are no Wolfram Alpha queries that can be used to solve the problem, such as Transform Shapes and Transformations on Functions. These categories do not show much change in the performance of both systems, as Wolfram Alpha cannot be used to provide answers for these categories. Incorporation of a more powerful tool, such as Python, could allow the system to successfully solve these problems.

We see a decrease in performance in the two categories dealing with inequalities. While the LLM+Solver system outputs functionally correct inequalities, it formats inequality signs in a manner that could not be accepted by Desmos, resulting in invalid graphs. This could be addressed with an extra instruction in the prompt explaining how to format inequalities. We also see poor performance in the Tangents to Parabolas category. In the case of questions asking for tangents through a point not on the parabola, Wolfram Alpha provides a correct result, but the accompanying step-by-step solution contains the answer to a different problem. As a result, the LLM+Solver system returns the incorrect answer. This issue has a more difficult solution, possibly requiring us to classify inputted problems and remove Wolfram's step-by-step solution in these cases.

5 Discussion

Overall, the results of this paper highlight the potential of tool use in combination with LLMs. By incorporating a solver into our LLM-based system, we were able to produce an automated graphing system with significantly higher accuracy. Through the design and development of an LLM+Solver system, a benchmark dataset based on Common Core, and an autoevaluation pipeline, we created an effective automated graphing system and the means to easily evaluate future iterations. The system has strong performance, and for many categories of problems it can consistently produce accurate outputs.

While the system is not yet fully reliable, there are many directions we could take in the future to improve it. For example, we could develop a classifier to distinguish between problem types, and make adjustments to the system for the individual problem categories. This individualized approach could improve accuracy for some categories as compared to our current, one-size-fits-all system. We also plan to evaluate our system using an open-source LLM instead of GPT-4, and make use of fine-tuning to improve the end-to-end latency of the system.

Acknowledgments and Disclosure of Funding

We would like to thank Ravi Kokku, Marc Pickett, Prasenjit Dey, Deepak Akkil and Paul Haley for helpful discussions and feedback.

References

[1] Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. A survey of chain of thought reasoning: Advances, frontiers and future, 2023.

[2] McKinsey & Company. What's the future of generative AI? an early view in 15 charts. `https://www.mckinsey.com/featured-insights/mckinsey-explainers/whats-the-future-of-generative-ai-an-early-view-in-15-charts`, 2023.

[3] PBC Desmos Studio. Desmos graphing calculator. `https://www.desmos.com/calculator`.

[4] Dermot Francis Donnelly-Hermosillo, Libby F. Gerard, and Marcia C. Linn. Impact of graph technologies in k-12 science and mathematics education. *Computers & Education*, 146:103748, 2020.

[5] Wolfram Alpha LLC. Wolfram alpha. `https://www.wolframalpha.com/`.

[6] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017.

[7] Council of Chief State School Officers National Governors Association Center for Best Practices. Common core state standards. `https://www.thecorestandards.org/Math/`, 2010.

[8] OpenAI. Gpt-4 technical report. `https://arxiv.org/abs/2303.08774`, 2023.

[9] Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models, 2023.

[10] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis, 2023.

[11] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023.

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

[13] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023.