Does GPT Really Get It? A Hierarchical Scale to Quantify Human and AI's Understanding of Algorithms

Mirabel Reid
School of Computer Science
Georgia Institute of Technology
mreid48@gatech.edu

Santosh S. Vempala
School of Computer Science
Georgia Institute of Technology
vempala@cc.gatech.edu

Abstract

As Large Language Models (LLMs) are used for increasingly complex cognitive tasks, a natural question is whether AI really *understands*. The study of understanding in LLMs is in its infancy, and the community has yet to incorporate research and insights from philosophy, psychology, and education. Here we focus on understanding *algorithms*, and propose a hierarchy of levels of understanding. We validate the hierarchy using a study with human subjects (undergraduate and graduate students). Following this, we apply the hierarchy to large language models (generations of GPT), revealing interesting similarities and differences with humans. We expect that our rigorous criteria for algorithm understanding will help monitor and quantify AI's progress in such cognitive domains.

1 Introduction

Since the release of GPT-4, mainstream users have begun to experiment with Large Language Models (LLMs) on increasingly complex tasks. However, the degree to which it is safe, legal, and ethical to rely on LLMs has been under fierce debate. Across many studies, researchers have identified apparent shortcomings of LLMs including hallucinations, inability to plan, and lack of understanding [Rawte et al., 2023, Mahowald et al., 2024, Valmeekam et al., 2023]. However, the literature notably lacks rigorous criteria to measure the progress toward solving these issues. A particular problem lies in claims surrounding understanding; AI understanding is frequently compared to human understanding, and it is folklore among AI researchers that the reasoning processes of LLMs differ from those of humans. While the concept of understanding is widely discussed, it remains ill-defined.

In this paper, we propose an *precise definition of understanding an algorithm* with the following properties: (a) it provides a scale by which to evaluate any entity's understanding of an algorithm, (b) it aligns with the standard usage of the term 'understanding' in philosophy and psychology, and (c) it can be used to evaluate AI's progress toward understanding algorithms.

Motivation - Why study algorithm understanding? Large language models are increasingly trusted for coding assistance. Code generation tools such as GitHub Copilot [GitHub, 2024] and Meta's Code Llama [Roziere et al., 2023] are currently used in practice to improve developer productivity [Vaithilingam et al., 2022, Mozannar et al., 2024] and assist novice programmers in learning [Kazemitabaar et al., 2023, Becker et al., 2023]. It is likely that the degree of AI involvement in software development will only grow as these tools improve. However, reliance on imperfect systems comes with risk. Tools such as Copilot are known to generate code that is subject to license [Becker et al., 2023] or contains security vulnerabilities [Pearce et al., 2022].

The question of whether LLMs demonstrate meaningful understanding of algorithms is relevant if we are relying on LLMs to implement algorithms in production or teach them to novice programmers.

NeurIPS 2024 Workshop on Behavioral Machine Learning.

Algorithm understanding is distinct from language understanding and deserves its own line of study. Those who argue that LLMs do not understand language draw a distinction between linguistic form and meaning[Bender and Koller, 2020, Mitchell and Krakauer, 2023, Pavlick, 2023]. When humans understand language, their understanding is informed by their communicative intent and the real-life properties of the objects described. Thus, a system trained only to replicate statistical correlations between words cannot understand language in the way that humans do. Algorithms, however, can be precisely represented using formal programming languages. One might argue that a computer can meaningfully observe an algorithm in full through code implementations and examples.

1.1 Related Work

Cognitive Abilities of LLMs. The past few years have seen an explosion of studies exploring the ability of LLMs to answer complex mathematical questions. Researchers have developed prompting strategies to enable multi-step reasoning Wei et al. [2022], Fu et al. [2022]. Still others fine-tune models to improve mathematical problem-solving Yu et al. [2023], Luo et al. [2023]. The benchmarks for these methods typically include large datasets such as GSM8k Cobbe et al. [2021] (grade school word problems) and MATH Hendrycks et al. [2021] (math competition problems). These works focus on correct evaluation and do not address whether the language models understand mathematical reasoning.

Others have studied metacognitive skills in LLMs. Didolkar et al. [2024] investigate whether LLMs can assign skill labels to mathematical problems. Also related is Aher et al. [2023] which proposes Turing experiments comparing humans and LLM simulations.

Understanding in LLMs. A parallel line of work investigates language understanding in LLMs. A key concept in the debate over language understanding is the difference between linguistic *form* and *meaning* Bender and Koller [2020], Merrill et al. [2021]. Bender and Koller [2020] argue that an AI trained only on linguistic form (i.e. text) cannot understand meaning. In an opinion piece, Pavlick [2023] counters this perspective, arguing that it is premature to draw conclusions on whether LLMs can model language understanding when the study of language models is itself in its infancy. There has been some effort to determine the extent to which LLMs represent linguistic meaning, primarily by studying word representations Li et al. [2021], Patel and Pavlick [2021]. For a survey on linguistic competence in LLMs, see Mahowald et al. [2024]. Also see Mitchell and Krakauer [2023] for a general survey on the debate over understanding.

1.2 Theories of Understanding

The debate over what constitutes understanding has a long history in philosophy and psychology. It is generally agreed that understanding is different from 'mere' knowledge, but the nature of that distinction is up for debate Pritchard [2009], Baumberger et al. [2016], Páez [2019]. Pritchard [2014] provides some examples of when the concepts of 'knowing why' and 'understanding why' may not overlap. Khalifa [2017] and Baumberger et al. [2016] are accessible surveys of this debate.

The philosophy of science also relates understanding and explanation, and the goal of explanation can be thought of as the production of understanding Friedman [1974], Grimm [2010], Baumberger et al. [2016]. Most theories assert that understanding a concept requires not only accurate knowledge of the concept, but also some additional mastery over it Baumberger et al. [2016].

For our hierarchy, we employ the framework of Understanding as Representation Manipulability (URM) Wilkenfeld [2013]. This theory posits that understanding arises from the ability to modify the internal representation of a concept in order to make effective inferences.

For language models, this representation is collected from the thousands of examples, explanations, and code snippets that appear in its training data. The mechanism behind human memory is not understood as precisely. However, humans also learn via hearing explanations, collecting examples, and reinforcing their knowledge. This forms a representation encoded in the neural pathways of our brains Durstewitz et al. [2000], György Buzsáki [2019]. The goal of our hierarchy will be to test how the existing internal representation can be manipulated to produce responses at different levels of difficulty.

2 A Definition of Understanding

We ask the question: *how well does an entity understand an algorithm?* Our goal is a definition of understanding that is itself algorithmically testable. Therefore, we adopt a functional lens, meaning that we define understanding by what it allows the entity to do.

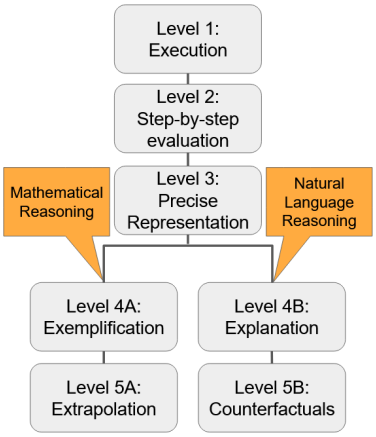


Figure 1: A hierarchy of understanding.

Levels of Understanding. In this section, we define understanding as a spectrum by presenting a series of levels. Understanding at each level is intended to be more difficult than the previous one, although they do not formally follow each other. Rather, they measure increasing levels of abstraction. Precise definitions of the levels are given in the appendix.

These levels of understanding and the partial ordering among them is summarized in Fig. 1. The first three levels measure the ability of the entity to recall a procedure and execute a known set of instructions. In Mayer's taxonomy of learning, they fall under the cognitive processes of recognizing, recalling, and executing [Mayer, 2002].

At the first level (denoted **Level 1**), the entity is capable of evaluating the algorithm on some 'simple' examples, where the simplicity of an input is defined by the length of the execution path. At this level, the entity has some representation of the input-output mapping, whether or not it can formally express it.

At the second level, the entity can describe how it evaluates the algorithm in a language that it knows. Level 2 requires the entity to output the execution steps of the algorithm on x as well as produce the correct answer.

The third level takes this one step further, requiring the entity to produce a set of instructions that can be followed to produce the right answer for any input x.

The next two levels target deep learning, and require a more complex manipulation of knowledge. We split the next levels into two subtrees, to distinguish cognitive processes utilizing functional linguistic skills from those utilizing mathematical reasoning [Mahowald et al., 2024].

At Level 4, the entity demonstrates an understanding of 'why' the algorithm is constructed as it is. It requires them to provide an example to illustrate a property (mathematical reasoning) or explain the existence of a property to a specified audience (natural language).

At Level 5, the entity can reason on perturbations of the algorithm and perturbations of the input, and it can describe the effect on the execution path. Under mathematical reasoning, this includes skills such as certifying if a modification to an algorithm changes the output for a subset of examples. Under natural language reasoning, this includes describing the effects of modifying inputs, or answering counterfactual questions about modifications to the algorithm.

3 Hypotheses and Methods

We conducted an experiment on LLMs and human participants with two main goals; 1) to assess the proposed hierarchical scale (Figure 1) as a tool for comparing levels of understanding, and 2) to rate algorithm understanding across generations of GPT. We will assess the scale with a student survey, where we can use self-reported understanding and educational level as a basis for comparison. Then, we will apply the same questions to GPT and assess its understanding on the same scale. Related to these goals, we test the following hypotheses:
1. The understanding hierarchy (Figure 1) captures depth of understanding.
We expect the fraction of correct answers to be non-increasing with higher levels of understanding. Furthermore, more education and training in algorithms should be reflected in the scores, so we expect graduate students to perform better than undergraduates.
2. Newer generations of GPT understand algorithms at a higher level than older generations.
Concretely, we expect an increase in performance at higher levels between GPT-3.5 and GPT-4.
3. LLMs will exhibit a performance gap between natural language reasoning and mathematical

reasoning tasks.
We expect the difference in performance between these two types of tasks to be much smaller in students than LLMs. Furthermore, we expect that GPT may have a higher performance at Level 3, since GPT is fine-tuned on code generation and has been exposed to code for common algorithms.

We use two classical algorithms to test our scale of understanding: the *Euclidean* algorithm for computing the greatest common divisor of two integers, and the *Ford-Fulkerson* algorithm for computing the maximum flow between two nodes on a directed graph with capacity constraints [Ford and Fulkerson, 1956]. Both algorithms are widely taught in undergraduate computer science curricula.

For each of the assessed algorithms, we produced a series of questions corresponding to each of the levels (Figure 1). Human participants were assigned an algorithm at random, and given five questions, testing either the mathematical or natural language reasoning branch.

The number of participants in the survey was $n = 34$ (10 doctoral and 24 undergraduate). Students who reported that they did not understand the algorithm or completed less than half of the survey questions were removed the analysis. This left $n = 23$ students (10 doctoral and 13 undergraduate). Of these students, ten had some teaching assistant experience in algorithms classes. For experiments on GPT, each version was prompted with 30 randomized questions per algorithm and per question number. Further details about the survey design will be deferred to the Appendix, where the full survey questions will also be available.

4 Results

Hypothesis 1. *The understanding hierarchy (Figure 1) captures depth of understanding.*

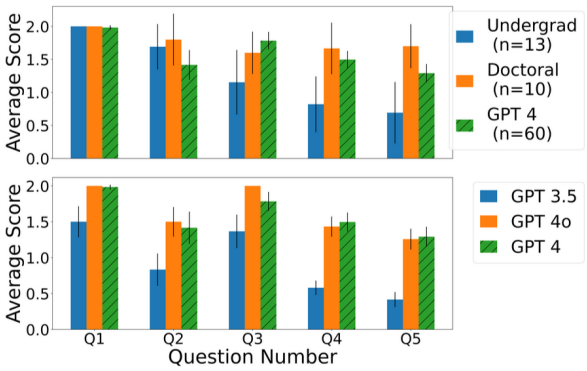


Figure 2: Mean scores between students (top) and GPT (bottom). Error bars are 95% confidence intervals.

We compare undergraduate and doctoral levels in Fig. 2. Across all students, the accuracy on the questions was highest for Question 1, and decreased uniformly through Question 5. Doctoral students performed better on average than undergraduates ($p < 0.05$). They received higher scores on Q4 and Q5 ($p < 0.05$), while the differences on Q1, Q2, and Q3 were not statistically significant.

Hypothesis 2. *Newer generations of GPT understand algorithms at a higher level.*
Among versions of GPT, GPT-4 and GPT-4o performed about the same, and the differences in their overall scores were not significant (Figure 2). Both GPT-4 and GPT-4o demonstrated an increase in score on every question compared to GPT-3.5 ($p < 0.05$).

The response score of GPT-4 was close to that of graduate students, as shown in Figure 2. Doctoral students scored better than GPT-4 on the extension questions (Q5) to a statistically significant degree. LLMs on average out-performed the undergraduate students on questions 3, 4, and 5 ($p < 0.05$), while the differences on Q1 and Q2 were not statistically significant.

Hypothesis 3. *LLMs will exhibit a performance gap between language and math reasoning.*
As shown in Figure 3, all three GPT versions tested performed better on language tasks than on math reasoning tasks for Ford-Fulkerson (significant with $p < 0.05$) despite student performance being the same or slightly worse. For GCD, all GPT versions performed better on language tasks on Level 4, but the same or slightly worse on Level 5.

We also hypothesized that the performance on code tasks would be higher compared to the performance on evaluation and reasoning tasks. We find that this does hold. As shown in Figure 2, LLM performed better on the coding tasks (Q3) than on the evaluation tasks (Q2), while the students exhibited the opposite trend.

Prompting with examples. We also investigated whether the use of example responses can improve the responses to MaxFlow problems. We tested 200 randomly instantiated MaxFlow problems (100 trivial and 100 intermediate), with and without a correct example response included in chat history.

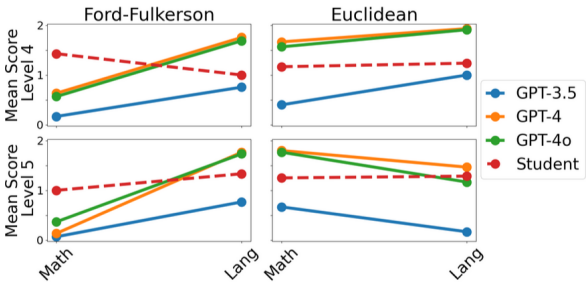


Figure 3: The difference in mean score between mathematical and language reasoning tasks on Ford-Fulkerson (Left) and the Euclidean algorithm (right), on Level 4 tasks (top) and Level 5 tasks (bottom)

We find that this strategy generally causes the response to mimic the structure of the example response. As shown in Figure 4, including an example response improves accuracy for GPT-3.5, but has little to no effect on GPT-4o, and marginally decreases the accuracy for GPT-4. One possible explanation for this phenomenon is that GPT-4 naturally responds to the prompt with effective chain of thought reasoning; therefore, instructing it to reason in a specific format does not improve its reasoning abilities, and may in fact interfere with them.

Discussion. We have presented a hierarchical scale for quantifying the understanding of algorithms. We verified its predictions empirically on human subjects and used it to compare generations of GPT and students. Our results show a significant improvement from GPT-3.5 to GPT-4/4o at all levels of algorithm understanding.

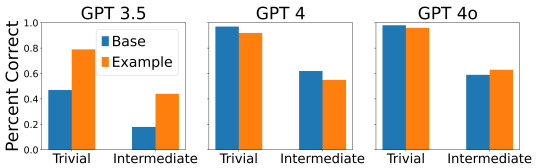


Figure 4: Accuracy of versions of GPT on MaxFlow problems. Shows base accuracy (blue) and accuracy with example (orange).

All versions of GPT were nearly perfect on code generation tasks. In contrast, students performed better at evaluating the algorithm than producing code on average. One possible reason for this is that the code for common algorithms, such as those tested, are prevalent in GPT's training data. Code is highly structured, so even if the particular implementation of the algorithm has not been observed by GPT, it could replicate the changes by statistical inference.

Another trend is that GPT generally performed better on language reasoning than mathematical reasoning, while student performance was about the same. This difference goes beyond algebraic computations - GPT struggles with questions testing common-sense graph reasoning that humans can answer easily. However, for explanation questions and reasoning questions that do not involve examples, GPT-4 and 4o give consistently quality responses. This may suggest that mathematical examples play a larger role in human understanding than in LLMs.

Our results show that the hierarchy of understanding is consistent with classical notions of depth of understanding when tested on humans. While the results are also consistent with later versions of GPT having a 'better' understanding of the tested algorithms than undergraduates, such a conclusion does not follow. We worked with a limited population size, and the difference is confounded by other factors such as subject fatigue. Further research is needed to compare the quality of GPT and human responses to questions about algorithms. Despite these limitations, we feel that our scale makes progress towards a testable definition of understanding and can be extended to other algorithmic and similarly precise realms of understanding.

Acknowledgements. The authors are grateful to Rosa Arriaga, Adam Kalai and Sashank Varma for helpful discussions. This work was funded in part by NSF Award CCF-2106444 and a Simons Investigator award.

References

Gati V Aher, Rosa I Arriaga, and Adam Tauman Kalai. Using large language models to simulate multiple humans and replicate human subject studies. In *International Conference on Machine*

Learning, pages 337–371. PMLR, 2023.

Christoph Baumberger, Claus Beisbart, and Georg Brun. What is understanding? an overview of recent debates in epistemology and philosophy of science. *Explaining understanding*, pages 1–34, 2016.

Brett A Becker, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, James Prather, and Eddie Antonio Santos. Programming is hard-or at least it used to be: Educational opportunities and challenges of ai code generation. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, pages 500–506, 2023.

Emily M Bender and Alexander Koller. Climbing towards nlu: On meaning, form, and understanding in the age of data. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pages 5185–5198, 2020.

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

Aniket Didolkar, Anirudh Goyal, Nan Rosemary Ke, Siyuan Guo, Michal Valko, Timothy Lillicrap, Danilo Rezende, Yoshua Bengio, Michael Mozer, and Sanjeev Arora. Metacognitive capabilities of llms: An exploration in mathematical problem solving. *arXiv preprint arXiv:2405.12205*, 2024.

Daniel Durstewitz, Jeremy K Seamans, and Terrence J Sejnowski. Neurocomputational models of working memory. *Nature neuroscience*, 3(11):1184–1191, 2000.

Lester Randolph Ford and Delbert R Fulkerson. Maximal flow through a network. *Canadian journal of Mathematics*, 8:399–404, 1956.

Michael Friedman. Explanation and scientific understanding. *the Journal of Philosophy*, 71(1):5–19, 1974.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. Complexity-based prompting for multi-step reasoning. In *The Eleventh International Conference on Learning Representations*, 2022.

GitHub. Github copilot · your ai pair programmer", 2024. URL `https://github.com/features/copilot`.

Stephen R Grimm. The goal of explanation. *Studies in History and Philosophy of Science Part A*, 41 (4):337–344, 2010.

MD György Buzsáki. *The brain from inside out*. Oxford University Press, 2019.

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

Majeed Kazemitabaar, Justin Chow, Carl Ka To Ma, Barbara J Ericson, David Weintrop, and Tovi Grossman. Studying the effect of ai code generators on supporting novice learners in introductory programming. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–23, 2023.

Kareem Khalifa. *Understanding, explanation, and scientific knowledge*. Cambridge University Press, 2017.

Belinda Z Li, Maxwell Nye, and Jacob Andreas. Implicit representations of meaning in neural language models. *arXiv preprint arXiv:2106.00737*, 2021.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*, 2023.

Kyle Mahowald, Anna A Ivanova, Idan A Blank, Nancy Kanwisher, Joshua B Tenenbaum, and Evelina Fedorenko. Dissociating language and thought in large language models. *Trends in Cognitive Sciences*, 2024.

Richard E Mayer. Rote versus meaningful learning. *Theory into practice*, 41(4):226–232, 2002.

William Merrill, Yoav Goldberg, Roy Schwartz, and Noah A Smith. Provable limitations of acquiring meaning from ungrounded form: What will future language models understand? *Transactions of the Association for Computational Linguistics*, 9:1047–1060, 2021.

Melanie Mitchell and David C Krakauer. The debate over understanding in ai's large language models. *Proceedings of the National Academy of Sciences*, 120(13):e2215907120, 2023.

Brent Mittelstadt, Chris Russell, and Sandra Wachter. Explaining explanations in ai. In *Proceedings of the conference on fairness, accountability, and transparency*, pages 279–288, 2019.

Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. Reading between the lines: Modeling user behavior and costs in ai-assisted programming. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2024.

Andrés Páez. The pragmatic turn in explainable artificial intelligence (xai). *Minds and Machines*, 29 (3):441–459, 2019.

Roma Patel and Ellie Pavlick. Mapping language models to grounded conceptual spaces. In *International conference on learning representations*, 2021.

Ellie Pavlick. Symbols and grounding in large language models. *Philosophical Transactions of the Royal Society A*, 381(2251):20220041, 2023.

Hammond Pearce, Baleegh Ahmad, Benjamin Tan, Brendan Dolan-Gavitt, and Ramesh Karri. Asleep at the keyboard? assessing the security of github copilot's code contributions. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 754–768. IEEE, 2022.

Duncan Pritchard. Knowledge, understanding and epistemic value. *Royal Institute of Philosophy Supplements*, 64:19–43, 2009.

Duncan Pritchard. Knowledge and understanding. In *Virtue epistemology naturalized: Bridges between virtue epistemology and philosophy of science*, pages 315–327. Springer, 2014.

Vipula Rawte, Amit Sheth, and Amitava Das. A survey of hallucination in large foundation models. *arXiv preprint arXiv:2309.05922*, 2023.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.

Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*, pages 1–7, 2022.

Karthik Valmeekam, Matthew Marquez, Sarath Sreedharan, and Subbarao Kambhampati. On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*, 36:75993–76005, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Daniel A Wilkenfeld. Understanding as representation manipulability. *Synthese*, 190:997–1016, 2013.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

A Levels of Understanding: Full Definitions

In this section, we present precise definitions of the levels of understanding given in the main body of the paper. For concreteness, we also give an example of a question which could be answered by an entity that understands the Euclidean algorithm at each level. For simplicity, we present these levels as deterministic; however, they can be defined with a failure probability dependent on the entity's internal randomness and the required memory and time.

The *execution path* of $\mathcal{A}$ on an input x is the sequence of states taken by the algorithm when executing on x. The *trace* of the execution is the execution path plus the contents of the tape at each step. Finally, define a *property* to be a function mapping the trace or execution path to $\{0, 1\}$.

At the first level (denoted **Level 1**), the entity is capable of evaluating the algorithm on some 'simple' examples, where the simplicity of an input is defined by the length of the execution path. At this level, the entity has some representation of the input-output mapping, whether or not it can formally express it.

Definition (Level 1: Execution)**.** *$\mathcal{E}$ understands $\mathcal{A}$ at Level 1 if there exists parameters M_0, T_0 such that the following holds: for any $x \in \Omega$ with $M(x) \leq M_0$ and $T(x) \leq T_0$, $\mathcal{A}_{\mathcal{E}}(x) = f(x)$.*
Example: *Compute GCD(24, 15).*

At the next level, the entity can describe how it evaluates $f(x)$ in a language that it knows. Level 2 requires the entity to output the execution steps of the algorithm on x as well as produce the correct answer.

Definition (Level 2: Step-By-Step Evaluation)**.** *$\mathcal{E}$ understands $\mathcal{A}$ at Level 2 if, given an $x \in \Omega$ with $M(x) \leq M_{\mathcal{E}}$ it can provide one of the following:*

- *the execution path in natural language or code*
- *a flow chart or other unambiguous pictoral representation of the execution path*

executed when running $\mathcal{A}$ on x.
Example: *Compute GCD(462, 948) and show each step of the calculation.*

The next level will take this one step further, requiring the entity to produce a set of instructions that can be followed to produce the right answer for any $x \in \Omega$.

Definition (Level 3: Representation)**.** *$\mathcal{E}$ understands $\mathcal{A}$ at Level 3 if it understands at Levels 1 and 2, and it can produce one of the following:*

- *a formal representation; e.g., code for $\mathcal{A}$ in a Turing-complete programming language it knows, a structured natural language description, an abstract syntax tree or Turing machine diagram.*
- *an unambiguous description of the execution steps in natural language.*

Example: *Write a function in a programming language you know that can compute the GCD of any two integers.*

The first three levels measure the ability of the entity to recall a procedure and execute a known set of instructions. We place these in the category of 'shallow learning'; in Mayer's taxonomy, they fall under the cognitive processes of recognizing, recalling, and executing [Mayer, 2002]. Note that all three levels could be achieved by a hard-coded script.

The next two levels target deep learning, and measure cognitive processes in the 'Understanding' and 'Analyzing' categories. We split the next levels into two subtrees, to distinguish cognitive processes utilizing functional linguistic skills from those utilizing mathematical reasoning [Mahowald et al., 2024].

At Level 4, the entity demonstrates an understanding of 'why' the algorithm is constructed as it is.

Definition (Level 4a: Exemplification)**.** *Given a property P of an execution path of the algorithm $\mathcal{A}$, $\mathcal{E}$ can generate an $x \in \Omega$ which satisfies P or report that none exists.* ***Example:*** *Give an integer $0 < x < 55$ that requires the greatest number of recursive steps to compute GCD$(55, x)$. Describe how you chose this number.*

Definition (Level 4b: Explanation)**.** *Given $\mathcal{A}$, or a property P satisfied by the execution path of $\mathcal{A}(x)$, and an audience $\mathcal{E}'$, the entity can produce a text in natural language that has the following characteristics:*

- *Accurately describes the steps of the algorithm/execution path.*
- *Abstracts or shortens the full description by referencing other algorithms known by the audience.*
- *Uses examples and analogies to other algorithms known by the audience to convey intuition.*

Example: *You are teaching a student who understands basic math operations but struggles with algebra and division with remainders. Explain how the Euclidean algorithm is used to find the greatest common divisor (GCD) of two given numbers, prioritizing intuition.*

At Level 5, the entity can reason on perturbations of the algorithm and perturbations of the input, and it can describe the effect on the execution path.

Definition (Level 5a: Extrapolation)**.** *The entity can answer questions about $\mathcal{A}$ of the following form.*

- *Given an algorithm $\mathcal{A}'$, the entity can determine whether $\mathcal{A}$ and $\mathcal{A}'$ produce the same output on all $x \in \Omega$. If not, it can find a counterexample such that $\mathcal{A}'(x) \neq \mathcal{A}(x)$.*
- *Given a relation $R \subset \Omega \times \Omega$, the entity can find a pair $(x, x') \in R$ with different execution paths on $\mathcal{A}$.*

Example: *Determine whether the following statement is true. If not, provide a counterexample. If $x > y$, then computing $GCD(2x, y)$ with the Euclidean algorithm requires more division operations than computing $GCD(x, y)$.*

Definition (Level 5b: Counterfactual Reasoning)**.** *The entity can produce natural language descriptions of $\mathcal{A}$ of the following form.*

- *Given an algorithm $\mathcal{A}'$ and an audience $\mathcal{E}'$, the entity can produce an explanation (c.f. Level 4b) contrasting the two algorithms.*
- *Given a relation $R \subset \Omega \times \Omega$, the entity can describe a property highlighting the differences in execution paths for $(x, x') \in R$.*

Example: *Consider the Fibonacci sequence defined by $F(0) = 0$, $F(1) = 1$, and $F(n) = F(n-1) + F(n-2)$ for $n \geq 2$. Why do consecutive Fibonacci numbers result in the maximum number of iterations for the Euclidean algorithm?*

These levels of understanding and the partial ordering among them is summarized in Fig. 1.

B Experimental Design

B.1 Evaluation

Each question is rated on a scale from zero to two. With the exception of the explanation questions, the scores have the following interpretations: (0) incorrect; (1) partially correct, surface level; (2) completely correct, thorough.

B.1.1 Evaluating Explanations

The quality of explanations and summaries can be subjective; however, they offer a deep insight into the subject's understanding of the material. We evaluate the explanations on three axes.

1. Correctness; the explanation is accurate and includes the key ideas of the algorithm.
2. Audience adaptation; the explanation is tuned to the audience, and the level of detail matches their prior knowledge.
3. Intuitiveness; the explanation conveys intuition; via contrast, example, analogy etc. and uses clear language.

Summaries and explanations are by definition *selective* and not necessarily complete [Mittelstadt et al., 2019]. The ability to identify key ideas is part of what differentiates explanation (Level 4) from the production of instructions (Level 3). An explanation is awarded 2/3 of a point for each bullet, for a maximum score of 2 per question.

B.2 Survey Questions

We conducted a survey on students of algorithms courses at a premier CS-teaching university. Each student was assigned either the Euclidean or Ford-Fulkerson Algorithm at random, and was asked to rate their own understanding of the algorithm on a six-point scale. Each survey consisted of five test questions to test their understanding. There were three versions of the survey for each algorithm, assigned at random. The questions are available in the Appendix.

This section contains the text of the survey given to human subjects. Each respondent was asked some preliminary questions about their course level and experience with algorithms. They were assigned either the Ford-Fulkerson algorithm or the Euclidean algorithm at random and asked to self-report their understanding (Figure 5). Some students who reported that they did not understand the Ford-Fulkerson algorithm were then given the survey for the Euclidean algorithm.

The student was then randomly assigned one of three survey versions, shown below.

B.3 Euclidean Algorithm

B.3.1 Version 1

1. Compute GCD(24, 15), and show each step of the algorithm.
2. Compute GCD(462, 946), and show each step of the algorithm.
3. Using a programming language you are familiar with, write code for a function gcd(a,b), which computes the greatest common divisor of two integers a and b using the Euclidean algorithm.
4. You are instructing a student in an algebra course. The student is familiar with basic mathematical operations such as addition, subtraction, multiplication and division. However, she struggles with algebraic equations and division with remainders. Explain the Euclidean algorithm to her, prioritizing conveying intuition.
5. Suppose gcd(a,b) = x and gcd(b,c) = y. Does gcd(a,b,c) = gcd(x,y)? Explain your reasoning.

B.3.2 Version 2

1. Compute GCD(24, 15), and show each step of the algorithm.
2. Compute GCD(4088, 1241), and show each step of the algorithm.
3. Using a programming language you are familiar with, write code for a function gcd(a,b), which computes the greatest common divisor of two integers a and b using the Euclidean algorithm
4. Consider computing GCD(55, x) for an input x. Give an integer $0 < x < 55$ that requires the greatest number of recursive steps to compute GCD(55, x). Describe how you chose this number.
5. Determine whether the following statement is true. If not, provide a counterexample. For any two positive, nonzero integers a,b, the equation sa + tb = GCD(a,b) has exactly one solution where s and t are integers.

B.3.3 Version 3

1. Compute GCD(24, 15), and show each step of the algorithm.
2. Compute GCD(1008, 468), and show each step of the algorithm.
3. Using a programming language you are familiar with, write code for a function gcd(a,b), which computes the greatest common divisor of two integers a and b using the Euclidean algorithm.

Rate your understanding of the Euclidean Algorithm to compute the greatest common divisor of two integers. Check the option that you feel applies best to you.

- I have never heard of this algorithm.
- I have heard of the algorithm, but do not know it.
- I know the algorithm, but do not understand it.
- I know the algorithm and understand it a little bit.
- I know the algorithm and have a fair understanding of it.
- I understand the algorithm completely.

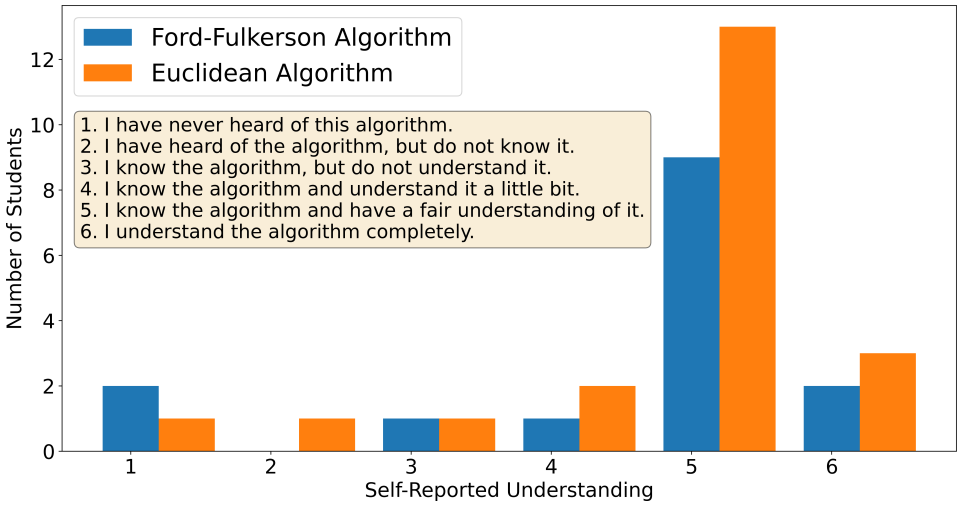


Figure 5: The six point-scale shown to students to self-report their understanding, and the distribution of responses.

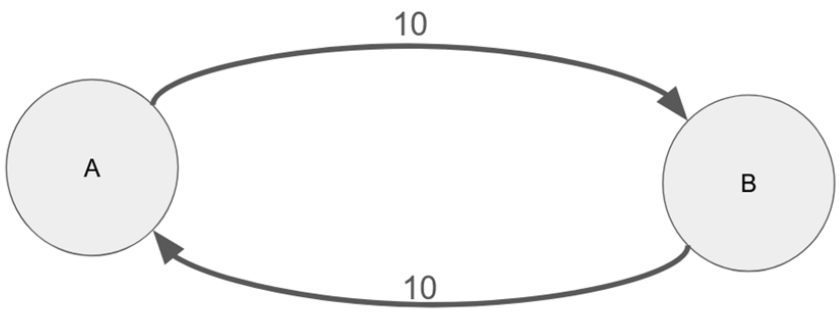
4. You are speaking with a mathematics student who understands modular arithmetic. Explain the proof that the Euclidean algorithm finds the greatest common divisor of two numbers.

5. The Fibonacci numbers are a sequence F(n) where F(0) = 0, F(1) = 1, and F(n) = F(n-1) + F(n-2) for any $n \geq 2$. Consecutive Fibonacci numbers can be thought of as 'worst case' inputs for the Euclidean algorithm. Can you explain why?

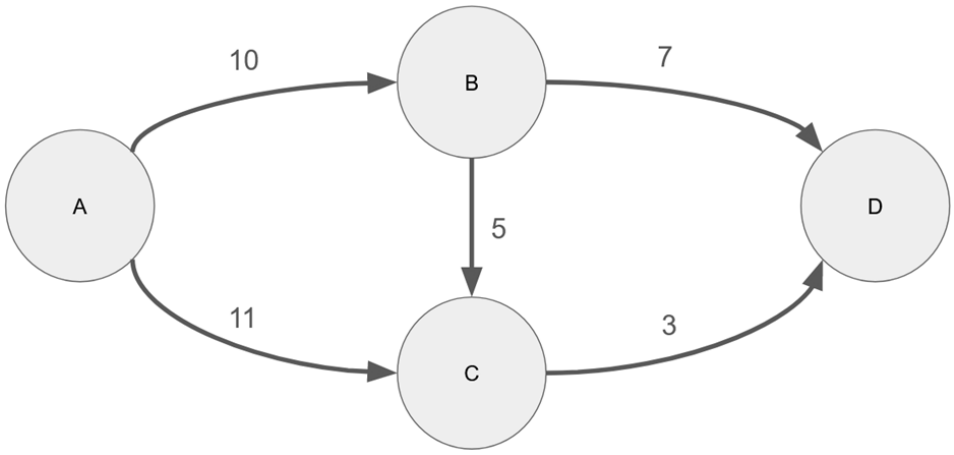
B.4 Ford-Fulkerson Algorithm

B.4.1 Version 1

1. Compute the maximum flow between A and B. List each augmenting path and the flow along the path at each step.



2. Compute the maximum flow between A and D. List each augmenting path and the flow along the path at each step.



3. The following is an implementation of breadth-first search in Python with one line missing. Write a condition to replace the highlighted text. If you are not familiar with Python syntax, you may use your best guess.

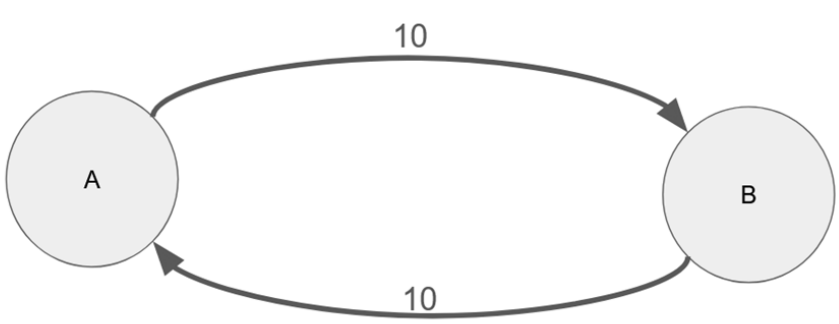
```python
def BFS(G, s, t):
    visited = [False]*G.shape[0]
    visited[s] = True
    q = [s]
    parents = [-1]*G.shape[0]
    while len(q) > 0:
        current = q[0]
        q = q[1:]
        for index, capacity in enumerate(G[current]):
            if <CONDITION>:
                q.append(index)
                visited[index] = True
                parents[index] = current
                if index == t:
                    return parents
    #If a path from s to t was not found, return None
    return None
```

4. Give an example of a graph where the Ford-Fulkerson algorithm computes exactly six augmenting flows before terminating. Write the example as a list of edges and capacities.

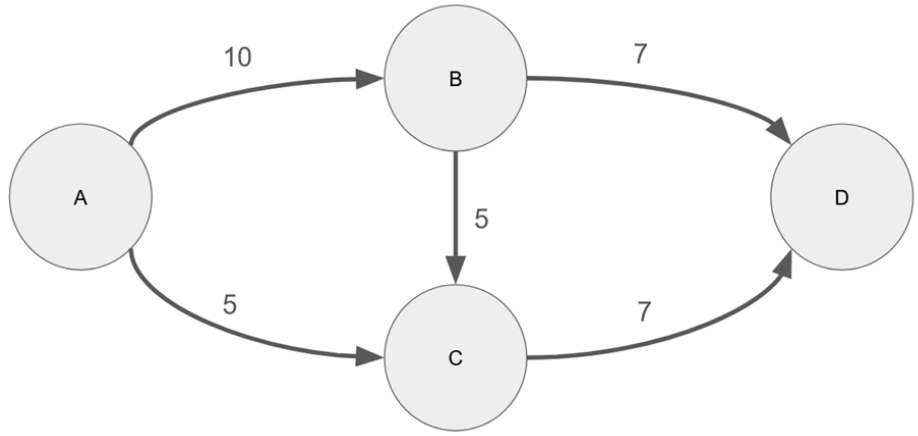
5. Consider an implementation of Ford-Fulkerson which uses a breadth-first search to find the augmenting path. Give an example which illustrates why the algorithm needs to track residual capacity of reverse edges.

B.4.2 Version 2

1. Compute the maximum flow between A and B. List each augmenting path and the flow along the path at each step.



2. Compute the maximum flow between A and D. List each augmenting path and the flow along the path at each step.



3. The following is an implementation of Ford-Fulkerson in Python with two lines missing. Write code to replace the highlighted text. If you are not familiar with Python syntax, you may use your best guess.

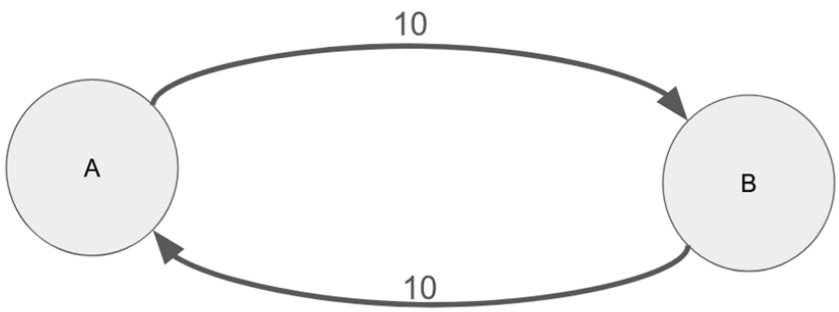
```
def FF(G,s,t):
    parents = BFS(G, s, t)
    max_flow = 0
    while parents is not None:
        flow_to_send = np.inf
        current = t
        while current != s:
            flow_to_send = min([flow_to_send, G[parents[current], current]])
            current = parents[current]

        current = t
        while current != s:
            <INSERT CODE HERE>
        max_flow +=flow_to_send

        parents = BFS(G, s, t)
    return max_flow
```

4. You are a civil engineer at a city planning commission. Your boss has asked you whether it is possible to calculate the maximum volume of traffic that can be routed between two destinations in the city. Describe the Ford-Fulkerson algorithm, but since he is a busy man, do not bore him with the details.

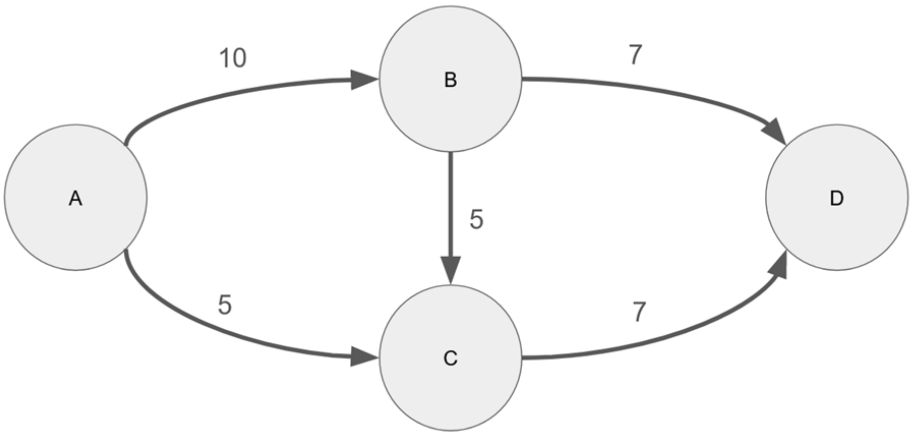
5. Suppose you are an event planner, and you need to determine the maximum amount of traffic that can be routed from the airport to one of two event venues. In other words, you want to compute the maximum flow between a source s and two sinks t1 and t2. How would you implement this?

B.4.3 Version 3

1. Compute the maximum flow between A and B. List each augmenting path and the flow along the path at each step.



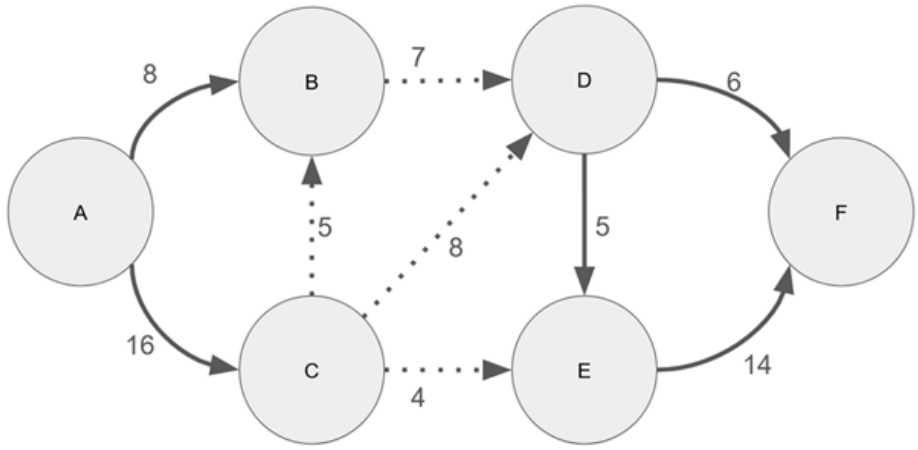
2. Compute the maximum flow between A and D. List each augmenting path and the flow along the path at each step.



3. The following is an implementation of Ford-Fulkerson in Python with three lines missing. Write code to replace the highlighted text. If you are not familiar with Python syntax, you may use your best guess.

```python
def FF(G,s,t):
    parents = BFS(G, s, t)
    max_flow = 0
    while parents is not None:
        flow_to_send = np.inf
        current = t
        while current != s:
            <INSERT CODE HERE>

        current = t
        while current != s:
            G[parents[current], current] -= flow_to_send
            G[current, parents[current]] += flow_to_send
            current = parents[current]
        max_flow +=flow_to_send

        parents = BFS(G, s, t)
    return max_flow
```

4. You are assisting a student in an algorithms course. He is struggling to understand the Ford-Fulkerson algorithm for computing MAXFLOW. Describe it to him, prioritizing conveying intuition.
5. An s-t cut is a set of edges which, when removed, divide vertex s and vertex t into separate components. For example, the dotted edges in the graph below are an A-F cut with capacity 24.



It turns out that the maximum s-t flow on a graph with capacities is equal to the minimum capacity of any s-t cut. Describe your intuition for why this might be true.

C Further details on LLM experiments

The LLM was queried using randomized versions of the survey. For evaluation questions, the input values were assigned uniformly at random within a given range. For the flow questions, the graph structure was also varied slightly. For the code questions, we took an example code implementation of the algorithm, randomly masked a line or group of lines, and asked the LLM to fill in the missing part. We also included several versions of the example, explanation, and extension questions.

Each survey was started in a fresh chat session, and the five questions were presented in the same order (within the survey, previous questions and responses were included in the chat history). In each experiment, the query included a system prompt to encourage the model to produce only relevant information. The system prompts used in the experiments are as follows:

> 'You will answer a series of questions related to computing the maximum flow on a directed graph. You provide concise responses and do not include detail or explanations unless explicitly requested by the user.'
> 'You will answer a series of questions related to the Euclidean algorithm for computing the greatest common divisor (gcd) of two integers. You provide concise responses and do not include detail or explanations unless explicitly requested by the user.'

GPT was queried via the OpenAI Chat Completions API using the default parameters (e.g. temperature=1). Specific versions queried are 'gpt-3.5-turbo-0125','gpt-4-turbo-2024-04-09', and 'gpt-4o-2024-05-13'.

We conducted two experiments with GPT. In the first, GPT was asked randomized versions of the questions given to students (a *quiz*), with the images being replaced with text descriptions of the graphs. For each quiz, the questions corresponding to each of the levels of understanding were asked in order, and the previous questions and responses were included in the API query. This quiz was repeated thirty times for each GPT version and algorithm. All trials were hand-graded on a scale from zero to two.

In the second experiment, GPT was queried with only the simple and intermediate evaluation questions for Max Flow (with varying graph structure and random weights). In one trial, GPT was first asked the simple question followed by the intermediate (with its response to the simple question passed to the API in the chat history). In the second trial, a sample response to an intermediate evaluation was passed to the chat history. The responses were checked by hand for correctness, and the rate of correctness was recorded.