Algorithmic Phase Transitions in Language Models: A Mechanistic Case Study of Arithmetic

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

The zero-shot capabilities of large language models make them powerful tools for solving a range of tasks without explicit training. However, it remains unclear how these models achieve such performance, or why they can zero-shot some tasks but not others. In this paper, we shed some light on this phenomenon by investigating algorithmic stability in language models: how perturbations in task specifications may change the problem-solving strategy employed by the model. While certain tasks may benefit from algorithmic instability (for example, sorting or searching under different assumptions), we focus on a task where algorithmic stability is needed: two-operand arithmetic. Surprisingly, even on this straightforward task, we find that `Gemma-2-2b` employings substantially different computational models on closely related subtasks. Our findings suggest that algorithmic instability may be a contributing factor to language models' poor zero-shot performance across many logical reasoning tasks, as they struggle to abstract different problem-solving strategies and smoothly transition between them.

1 Introduction

Language models have demonstrated remarkable capabilities across diverse benchmarks. However, it is unclear how the individual computational components in the model are working together to drive this performance [30, 25]. Specifically, for any given task, we do not understand the decision criteria of these models [5]. Identifying and explaining the computational mechanisms that drive model behavior would help us improve performance [18, 23], enhance interpretability [29, 9], and mitigate harmful their behavior [14].

In this paper, we focus on algorithmic phase transitions and stability. Across many natural tasks, we expect strong learners to exhibit both algorithmic stability and instability. For example, for two-operand addition, we expect a strong learner to add four-digit numbers in the same way as it adds eight-digit numbers. On the other hand, we may expect a set of competition math problems to be algorithmically unstable since solving them may require a range of techniques from counting to geometry. Intuitively, the reasoning of a generalizable model should be stable under perturbations across algorithmically stable problems and vice versa. ***Surprisingly, we find that language models exhibit algorithmic instability even on simple tasks such as arithmetic.***

For a given task, we first identify various algorithms implemented by the model to solve the task, then isolate sufficient conditions that either enable, or disable, transitions between them. Seminal works, such as [19, 31], have shown that Transformers [27] implement different algorithms in tandem to solve the same task. Additionally, the interactions between these parallel algorithms are highly sensitive to training and architectural hyperparameters. Similarly, [18] presented a case study of circuits within the same model reused to perform two distinct tasks. Our work builds off of these

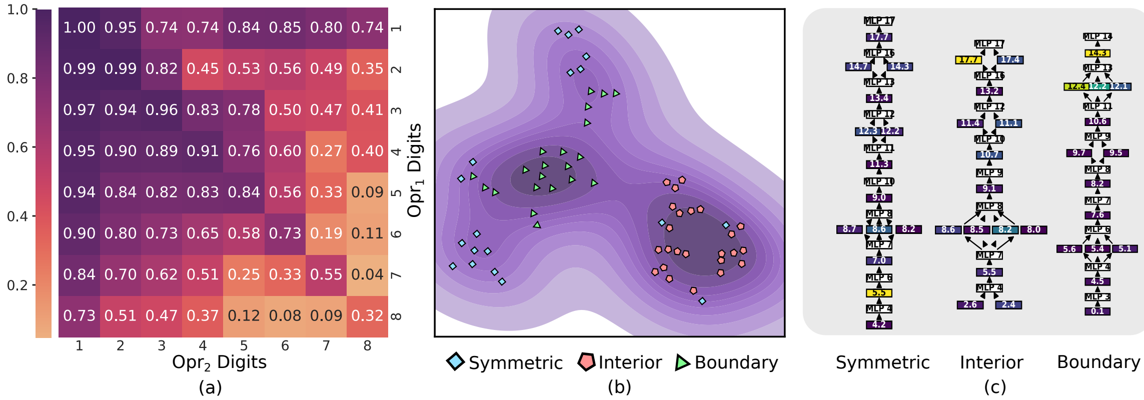

Figure 1: (a) Accuracy of `Gemma-2-2b` on two-operand addition as the number of digits in both operands vary. Accuracy is measured through exact string match. (b) t-SNE plot (with perplexity=3) where each point is the algorithm `Gemma-2-2b` implements for a m, n-digit addition problem. We capture algorithmic similarity measuring the importance of each attention head. (c) Sample circuits from $8, 1$-, $6, 3$-, and $4, 4$-digit addition, from left to right. These tasks are representative of the phases we identify: symmetry, interior, and boundary

results. For the same task, as complexity increases, we present preliminary evidence that shows models undergo periods of algorithmic stability and sharp phase transitions.

We illustrate this phenomena on arithmetic problems, specifically, two-operand addition. Arithmetic, especially where the operands have many digits, represents a complex task for language models [12, 9, 16, 22]. The task complexity may be controlled by changing the number of digits in each operand. And the task encompasses sufficiently rich algorithms, since there are many possible implementations that span a host of problem-solving strategies such as look-up table, divide-and-conquer, or dynamic programming.

A growing body of literature—mechanistic interpretability—isolates components of interest by first perturbing them and then observing their downstream effects on model behavior [20, 3, 31]. If removing a neuron collapses performance, then it is likely that neuron and its subsidiaries were necessary for the performance and vice versa [28, 17, 11]. After a collection of mechanisms are found, fine-grained interpretability techniques [6, 19, 1, 24, 2, 31, 18] are applied to generate a human-understandable explanation of how the computational mechanisms are operating together.

In sum, our contributions are threefold:

- We present evidence that for operand arithmetic, ***language models undergo sharp algorithmic phase transitions.*** This may provide a sufficient explanation for their lack of generalization across arithmetic tasks.
- We present ***novel methods and techniques to quantify these changes*** in contrast to many of the existing qualitative techinques that currently exist in mechanistic interpretability.
- We ***conjecture a preliminary framework to explain the source of these transitions.***

2 Identifying Phase Transitions Across Task Perturbations

All of our subsequent experiments are performed with `Gemma-2-2b` [26] on two-operand addition. We discuss generalizing our methodologies in Appendix C. Therein, we also present preliminary experimental results for these additional settings. We specifically choose arithmetic because we expect a model that performs well to learn general rules and algorithms. Intuitively, we expect any model which performs addition successfully to be algorithmically stable. That is, there is no change between how the model adds two four digit numbers and two eight digit numbers. Surprisingly, in the following subsections, we reveal this not to be the case for `Gemma-2-2b`. These findings suggest that algorithmic instability may be a contributing factor to language models' poor performance on many logical reasoning tasks, as they struggle to abstract different problem-solving strategies and smoothly transitions between them.

2.1 Preliminaries

We consider problems in the form of $a + b$ where $a, b \in \mathbb{Z}^+$ and $a, b \leq 10^9 - 1$. For the sake of brevity, we refer to a as Opr_1 and b as Opr_2. Let us denote the class of m, n-digit addition problems as the set $\{(a, b) : 10^{m-1} \leq a \leq 10^m - 1, 10^n \leq b \leq 10^n - 1\}$. In other words, $\mathrm{Opr}_1, \mathrm{Opr}_2$ have m, n digits, respectively. We aim to characterize the algorithm implemented by the model for any m, n-digit addition problem and then observe the differences as both m, n change. When the prompt the language model, on a given problem $a + b = ?$, we first pass it $k = 2$ few-shot examples from problems of the same class. We ablate over this hyperparameter, but observe no significant change in performance as k increases. For brevity, let us denote the number of digits in Opr_i as $\sharp\mathrm{Opr}_i$. In all of our experiments we vary the number of digits in each operand between one and eight.

2.2 Baseline Performance

To get a rough idea of what the model's performance looks like without any intervention, we benchmark `Gemma-2-2b` across all problem classes. For every problem class, we sample $n = 1000$ problems. These results are shown in Fig. 1 (a). We see that as the task complexity increases (as the number of digits in each operand increase), the performance of the model monotonically decreases. Moreover, the performance of the model is not symmetric. That is, the performance on $a + b$ is different from $b + a$ with a positive bias towards Opr_1 having more digits. We can see also see that the magnitude of asymmetry increases as the number of digits in both operands increase. Notice also that the boundaries of the matrix in Fig. 1 does not obey this property, in the sense that it's performance does not degrade as quickly as the number of digits in each operand increases, nor does it change as much as the number of digits change.

Proposed phases. ***Based on this initial observation, we propose three main phases defined by the drastic changes in performance.*** More specifically, we partition the set of all tasks into three sets: symmetric, boundary, interior. We say that a task is symmetric if $\sharp\mathrm{Opr}_1 = \sharp\mathrm{Opr}_2$. We say that a task is on the boundary if either $\sharp\mathrm{Opr}_1 = 1$ or $\sharp\mathrm{Opr}_2 = 1$. And lastly, a task on the interior is any task that does not fall into either of these categories. Across the set of tasks, we see that as the model transitions between these tasks, performance changes drastically. We then specifically reverse engineer the circuits from each of these categorizations and compare them structurally. The details on this operation can be found in the next section.

2.3 Reverse Engineering Arithmetic Circuits

To find the causal subcircuits responsible for the algorithmic behavior of the model, we use *activation patching*. Activation patching is the process through which we isolate components in the computational graph that is either necessary or sufficient for the task at hand. Since we are interested in the circuits for each task, we reverse engineer them independently. Let $\mathcal{D}_{n,m}$ be the set of all n, m-digit addition problems. Similar to the setup in [17, 29, 9, 18], we perform circuit denoising. Fix some $x \in \mathcal{D}_{n,m}$ and let y be the solution to this addition problem. We first perform a clean run: passing x through to the model and storing all of the intermediate activations. Then, we perform a corrupted run: sample some other $x' \in \mathcal{D}$ such that $x' \neq x$ which also has answer $y' \neq y$. In other words, we want to find an input such that its expected output should be different from the clean input.

Now that we have the activation caches for both the clean and corrupted run, to determine if a computational component in the model has a sufficient causal impact on the performance of the model, we start by running the corrupted input through the model and then at the component of interest, patching in the output of that component from the clean then. Then, for every subsequent computational component, we recompute its output. Let y^* denote the model prediction with the replaced components.

To quantitatively measure the impact of this computational component, we measure $\mathbb{P}(y^* = y|x') - \mathbb{P}(y^* = y|x)$. This is estimated by simply averaging the Softmax of the target tokens in the logit layer across all of the token indices that we are predicting. Note that there are many other methods for measuring this change in performance, and we discuss them in detail in Appendix A.

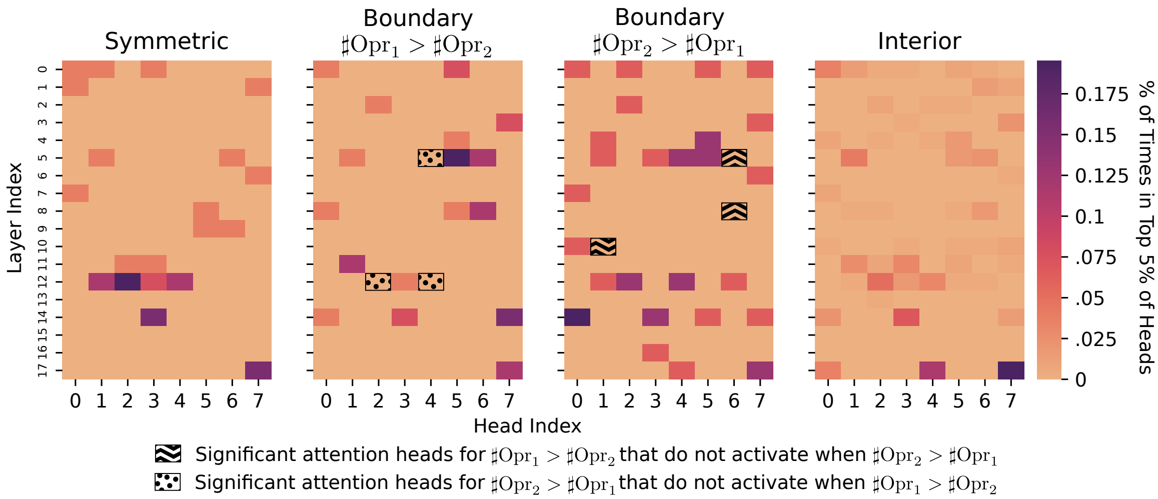

Figure 2: For each subtask, each heatmap represents the probability that that particular attention head will be in the top 5% of the most influential attention heads. We split the boundary subtask into two cases: where $\sharp\text{Opr}_1 > \sharp\text{Opr}_2$ and vice versa.

We only patch the outputs of the attention heads and do not touch any of the MLPs. This is because the prevailing sentiment is that the attention heads are responsible for any algorithmic manipulations of the tokens that the model is doing, whereas the MLPs are mainly for knowledge and fact retrieval [7, 8, 10]. Also, note that when we perform patching, we are patching the entire activation from the clean input into the model, there exists more granular methods of patching (for example, patching only a specific token in at an attention head) that we explore in the latter sections.

We perform activation patching over $n = 1000$ clean and counterfactual inputs for each task. The resulting circuits for all of the tasks can be found in Appendix E. We also present a detailed analysis of the different classes of circuits we find in the following subsections.

2.4 Circuit Stability Under Across Task Classes

Now that we have found the circuits, we want to see whether there is agreement in the set of components we have found within the proposed task classes we identify. As a result of activation patching, we have, for each attention head, the amount of performance we recover by patching in the clean output associated with it. Thus, using t-SNE we compare these heatmaps by projecting it into $\mathbb{R}^2$. The result of this is shown in Fig. 1 (b). We see that within the classes we identify: symmetric, interior, boundary. Algorithmic similar is generally conserved, but between the different classes the algorithms are drastically different. Some sample circuits within each of these classes are shown in Fig. 1 (c). The algorithmic differences between the tasks may be one explanation for the phase transition in performance as well.

We also plot the Pearson correlation between the influence activations between every pair of tasks. This is shown in Fig. 4. Through these plots demonstrate the existence of phase transitions across tasks for the model, it remains to characterize these phase algorithmically and reverse engineer the specific algorithms being applied in each of these phases. In this paper, we analyze the former and give some insight into how to achieve the latter based on our observations.

2.5 Characteristics of Phases

Herein, we characterize the different algorithmic classes we claim to have identify based on their circuit diagrams. Please see Appendix E to see all of the circuits that we discover. Note that to determine the underlying circuit, we look at the top 10% of all attention heads based on their influence score. We briefly discuss this hyperparameter and its effects on our results in Appendix B.

We find that all of the subcircuits are activating in layer 0, specifically attention head 0 and also in the last layer at attention head 7. We posit that this interaction is responsible for task identification and also writing the final answer to the output stream.

Symmetric. The $1,1$-digit addition circuit is different from all of the other circuits. This specific circuit is shown in the appendices. We hypothesize that this circuit is performing retrieval and memorization. Moreover, all of the algorithms for the phases that we identify below "call" this subcircuit as a recursive function.

We defer the analysis of $1,1$-digit addition to the following sections and herein analyze the circuits of n,n-digit addition where $n > 1$. It can be observed that most of the circuits that fall into this classification are top-heavy. That is, most of their influential attention heads lie in layers 17-14. Moreover, there are also activations concentrated about the middle layers of the network, mostly layers 5, 8. Specifically, we see that across all of the circuits attention heads 12.4, 12.2 and 17.7 are activating strongly. There are also early activations in the first layer across heads 0.0, 0.2, 0.3. Please refer to Fig. 6 to see all of the circuits associated with this subtask. This is shown in Fig. 2.

Boundary. In contrary to the circuits that we found in the symmetric subtasks, we found that there are generally two peaks of activations: in the early layers and in the latter layers. Specifically, when $\sharp\mathrm{Opr}_1 > \sharp\mathrm{Opr}_2$, we find that generally attention heads in layer 5 (5.5, 56) are strongly activated, as well as attention heads in layer 14 (14.3, 14.7). Similar to the circuits when $\sharp\mathrm{Opr}_1 = \sharp\mathrm{Opr}_2$, we see that attention heads in the early layers are strongly activated such as 2.2, as well as the same attention heads in the first layer of the previous circuits.

We also notice that within the boundary class as well, there seems to be two distinct behaviors. This is seen in Fig. 1. When $\sharp\mathrm{Opr}_1 > \sharp\mathrm{Opr}_2$ we see different circuits compared to $\sharp\mathrm{Opr}_2 > \sharp\mathrm{Opr}_1$. The subcircuits that correspond to these cases are further highlighted in Fig. 2. We find that much more attention heads are involved in the computation of this latter case compared to the former. Not only that, there are specific high impact attention heads in the middle layers that are disjoint from each other. These attention heads are highlighted in Fig. 2.

Interior. Lastly, circuits on the interior are generally diffuse through out the model. This can be observed in the right-most panel of Fig. 2. We find that in general many of the attention heads in the middle layer that are shared between the symmetric case and boundary cases can be found as subset of high impact attention heads in this subtask.

3 What's in a phase? One possible explanation

Though we have identified that phase transitions occur within the large langauge model, it remains to demonstrate what these stages actually are and the algorithmic differences between them rather than just identifying the computational difference.

Herein, we present our framework which we call *Dynamic Recursive Tooling*. We first demonstrate how for simple addition problems (when both operands only have 1 or 2 digits), LLMs directly memorize the answer. However, as the task complexity increases, for example when the digits of one of the operands increase, the LLM quickly transitions to a different new algorithm through a divide and conquer schema. Next, as the task complexity increases again by increasing the number of digits in both operands, we discover that the algorithm changes to one that resembles dynamic programming. We provide some preliminary experimental evidence for these stages and conjecture about the effect the architecture plays into these interactions.

Retrieval. Based on the substructures that we discover in the circuits where $\sharp\mathrm{Opr}_1 > 1$ or $\sharp\mathrm{Opr}_2 > 1$, we hypothesize that for $1,1$-digit addition, the model is storing the results of these as memories and then simply retrieving them. Then, for subsequent tasks that rely on these problems, the model recursively "calls" these components. This behavior is shown in the first panel of Fig. 3.

Bypass Retrieval. When either $\sharp\mathrm{Opr}_1 = 1$ or $\sharp\mathrm{Opr}_2 = 1$, we propose that the model performs *bypass retrieval*. That is, for any digit in the operand that is not in the ones or tens place, the model first separates these digits from the rest of the digits, then independently processes them. The ones and tens digits are added to the other one digit operand. This is done through retrieval. Let us denote the portion of digits that were separated out as the *bypassed* digits of the number. Once retrieval is complete, we hypothesize that the bypassed digits are joined together with the retrieval digits through a concatenation operation.

Recurisve Tooling. For $\sharp\text{Opr}_1 > 1$ and $\sharp\text{Opr}_2 > 1$, we hypothesize that the model is performing *recursive tooling*. Essentially, it is combining the techniques from the two previous situations together into one algorithm. First it is performing all single digit addition through direct retrieval. Then, it performs single digit retrieval again among the resultant, shifted values. This process is shown in the last panel of Fig. 3. In the appendix, we show some preliminary evidence for each of these stages and discuss detailed experimental set ups that lead us to these hypotheses.

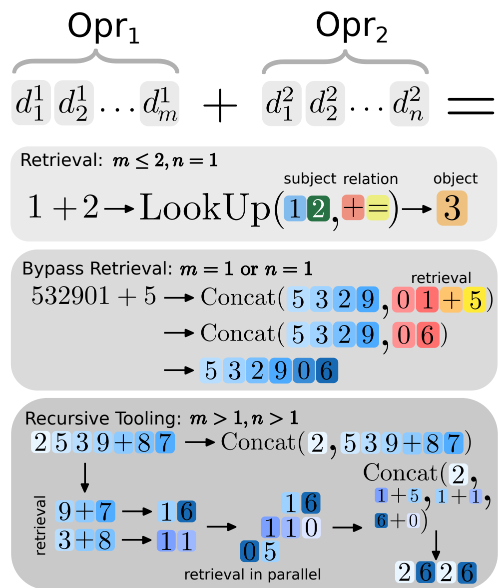

Figure 3: Our hypothesis for the mechanisms behind the different phases.

4 Discussion

In this paper, we explored algorithmic phase transitions induced by changes in task complexity. Surprisingly, we find that even on simple tasks such as arithemtic, language models exhibit sharp algorithmic phase transitions across different subtasks. Our work presents a novel method to operationalize and quantify this transition by identifying and comparing the structures of the underlying circuits driving these behaviors. Our work relies on a host of seminal techniques in mechanistic interpretability [29, 9, 21]. Specifically, we heavily rely on activation patching to identify these subcircuits. The limitations of activation patching are known and discussed extensively throughout the literature, however, they still remain the most effective methods to tractably identify subcircuits. One potential source of future work is to test the robustness of the subcircuits that we identified by perturbing the weights of the network itself. In this way, finding a subcircuit that is minimax-optimal.

Moreover, our work operationalizes algorithmic changes as structure changes. We are the first to propose clustering techniques to quantify these structural changes. However, there may exist other methods of quantifying this change. We leave this open and it could serve as a good basis for future work, especially in the mechanistic interpretability subfield.

Lastly, our results could have strong implications for improving and explaining the logical reasoning abilities of transformers in general. We demonstrated that even on this simple task, language models are unable to identify and learn generalizable algorithms. Perhaps this is a barrier for more complex reasonings as well as.

5 Conclusion

In this paper, we demonstrate that for two-operand addition, surprisingly, language models are not algorithmically stable. Specifically, the underlying algorithm that is used to solve the problem changes as the task is perturbed. Our results are the first to look at algorithmic stability with respect to task perturbations. We believe that this interpretability technique will allow researchers to better evaluate language model design choices. Moreover, it may also serve as a diagnostic tool to determine whether or not generalizing behavior is occuring. Specifically, we find that algorithmic instability correlates strongly with a lack of generalizability. We hope that these insights will bridge many of the toy examples in mechanistic interpretability to actual practitoiners.

References

[1] Lawrence Chan et al. "Causal scrubbing, a method for rigorously testing interpretability hypotheses". In: *AI Alignment Forum* (Dec. 2022).

[2] Bilal Chughtai, Lawrence Chan, and Neel Nanda. "A Toy Model of Universality: Reverse Engineering how Networks Learn Group Operations". In: *Proceedings of the 40th International Conference on Machine Learning*. May 2023. URL: https://arxiv.org/pdf/2302.03025.

[3] Arthur Conmy et al. "Towards Automated Circuit Discovery for Mechanistic Interpretability". In: *Thirty-Seventh Conference on Neural Information Processing Systems*. Oct. 2023. URL: https://arxiv.org/abs/2304.14997.

[4] Aniruddha Deb et al. *Fill in the Blank: Exploring and Enhancing LLM Capabilities for Backward Reasoning in Math Word Problems*. 2024. arXiv: 2310.01991 [cs.CL]. URL: https://arxiv.org/abs/2310.01991.

[5] Rudresh Dwivedi et al. "Explainable AI (XAI): Core ideas, techniques, and solutions". In: *ACM Computing Surveys* 55.9 (2023), pp. 1–33.

[6] Amir Feder et al. "CausaLM: Causal Model Explanation Through Counterfactual Language Models". In: *Computational Linguistics* 47.2 (June 2021). Place: Cambridge, MA Publisher: MIT Press, pp. 333–386. DOI: 10.1162/coli_a_00404. URL: https://aclanthology.org/2021.cl-2.13.

[7] Mor Geva et al. "Transformer Feed-Forward Layers Are Key-Value Memories". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Ed. by Marie-Francine Moens et al. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 5484–5495. DOI: 10.18653/v1/2021.emnlp-main.446. URL: https://aclanthology.org/2021.emnlp-main.446.

[8] Mor Geva et al. "Transformer Feed-Forward Layers Build Predictions by Promoting Concepts in the Vocabulary Space". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 30–45. DOI: 10.18653/v1/2022.emnlp-main.3. URL: https://aclanthology.org/2022.emnlp-main.3.

[9] Michael Hanna, Ollie Liu, and Alexandre Variengien. "How does GPT-2 compute greater-than?: Interpreting mathematical abilities in a pre-trained language model". In: *Thirty-seventh Conference on Neural Information Processing Systems*. May 2023. URL: https://openreview.net/forum?id=p4PckNQR8k.

[10] Adi Haviv et al. "Understanding Transformer Memorization Recall Through Idioms". In: *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*. Ed. by Andreas Vlachos and Isabelle Augenstein. Dubrovnik, Croatia: Association for Computational Linguistics, May 2023, pp. 248–264. DOI: 10.18653/v1/2023.eacl-main.19. URL: https://aclanthology.org/2023.eacl-main.19.

[11] Stefan Heimersheim and Neel Nanda. "How to use and interpret activation patching". In: *arXiv preprint arXiv:2404.15255* (2024).

[12] Dan Hendrycks et al. "Measuring Mathematical Problem Solving With the MATH Dataset". In: *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*. 2021. URL: https://openreview.net/forum?id=7Bywt2mQsCe.

[13] Dan Hendrycks et al. *Measuring Mathematical Problem Solving With the MATH Dataset*. 2021. arXiv: 2103.03874 [cs.LG]. URL: https://arxiv.org/abs/2103.03874.

[14] Andrew Lee et al. "A Mechanistic Understanding of Alignment Algorithms: A Case Study on DPO and Toxicity". In: *Forty-first International Conference on Machine Learning*. 2024. URL: https://openreview.net/forum?id=dBqHGZPGZI.

[15] Yujun Mao, Yoon Kim, and Yilun Zhou. *CHAMP: A Competition-level Dataset for Fine-Grained Analyses of LLMs' Mathematical Reasoning Capabilities*. 2024. arXiv: 2401.06961 [cs.CL]. URL: https://arxiv.org/abs/2401.06961.

[16] Yujun Mao, Yoon Kim, and Yilun Zhou. "WAMP: A Competition-Level Dataset for Assessing the Mathematical Reasoning Capabilities of LLMs". In: *The 3rd Workshop on Mathematical Reasoning and AI at NeurIPS'23*. 2023. URL: https://openreview.net/forum?id=loDIYsNLbw.

[17] Kevin Meng et al. "Locating and Editing Factual Associations in GPT". In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh et al. Oct. 2022. URL: https://openreview.net/forum?id=-h6WAS6eE4.

[18] Jack Merullo, Carsten Eickhoff, and Ellie Pavlick. "Circuit Component Reuse Across Tasks in Transformer Language Models". In: *The Twelfth International Conference on Learning Representations*. Sept. 2023. URL: https://openreview.net/forum?id=fpoAYV6Wsk.

[19] Neel Nanda et al. "Progress measures for grokking via mechanistic interpretability". In: *The Eleventh International Conference on Learning Representations*. Sept. 2022. URL: https://openreview.net/forum?id=9XFSbDPmdW.

[20] Catherine Olsson et al. "In-context learning and induction heads". In: *arXiv preprint arXiv:2209.11895* (2022).

[21] Alethea Power et al. *Grokking: Generalization Beyond Overfitting on Small Datasets*. Jan. 2022. URL: https://arxiv.org/abs/2201.02177.

[22] Luyu Qiu et al. "Dissecting Multiplication in Transformers: Insights into LLMs". In: *arXiv preprint arXiv:2407.15360* (2024).

[23] Gautam Reddy. "The mechanistic basis of data dependence and abrupt learning in an in-context classification task". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=aN4Jf6Cx69.

[24] Raanan Yehezkel Rohekar, Yaniv Gurwicz, and Shami Nisimov. "Causal Interpretation of Self-Attention in Pre-Trained Transformers". In: *Thirty-seventh Conference on Neural Information Processing Systems*. 2023. URL: https://openreview.net/forum?id=DS4rKySlYC.

[25] Rylan Schaeffer, Brando Miranda, and Sanmi Koyejo. "Are Emergent Abilities of Large Language Models a Mirage?" In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh et al. Vol. 36. Curran Associates, Inc., 2023, pp. 55565–55581. URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/adc98a266f45005c403b8311ca7e8bd7-Paper-Conference.pdf.

[26] Gemma Team et al. *Gemma 2: Improving Open Language Models at a Practical Size*. 2024. arXiv: 2408.00118 [cs.CL]. URL: https://arxiv.org/abs/2408.00118.

[27] Ashish Vaswani et al. "Attention is all you need". In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.

[28] Jesse Vig et al. "Investigating gender bias in language models using causal mediation analysis". In: *Advances in neural information processing systems*. Vol. 33. May 2020, pp. 12388–12401. URL: https://proceedings.neurips.cc/paper/2020/file/92650b2e92217715fe312e6fa7b90d82-Paper.pdf.

[29] Kevin Ro Wang et al. "Interpretability in the Wild: a Circuit for Indirect Object Identification in GPT-2 Small". In: *The Eleventh International Conference on Learning Representations*. Sept. 2022. URL: https://openreview.net/forum?id=NpsVSN6o4ul.

[30] Jason Wei et al. "Emergent Abilities of Large Language Models". In: *Transactions on Machine Learning Research* (2022). ISSN: 2835-8856. URL: https://openreview.net/forum?id=yzkSU5zdwD.

[31] Zhiqian Zhong et al. "The Clock and the Pizza: Two Stories in Mechanistic Explanation of Neural Networks". In: *Advances in Neural Information Processing Systems*. Nov. 2023. URL: https://arxiv.org/pdf/2306.17844.

A Activation Patching

Activation patching is a mechanistic interpretability technique used to understand how specific internal activations of a neural network contribute to its output. The process involves selecting two similar prompts, running the model with one prompt (the source) and saving its internal activations, then running the model with the second prompt (the destination) while overwriting selected activations within the second prompt with those from the source prompt. This allows us to observe how the model's behavior changes when certain activations are altered. Thus, we can identify which components of the model are responsible for specific outputs. There are generally two techniques of activation patching known as denoising (inserting clean activations into corrupted runs to see if they restore correct behavior) and noising (inserting corrupted activations into clean runs to see if they disrupt behavior). These techniques allow us to analyze circuits within neural networks and reveale whether certain activations are necessary or sufficient for specific behaviors.

B Addition Experiment

Much of the current work that's examining the accuracy of LLMs utilize word problems and much more complex math than simple arithmetic to evaluate their skills [4, 13, 15]. These studies rely both on an LLM's ability to interpret a word problem and its complex mathematical reasoning. The complex math problems implicitly require arithmetic in order to solve them correctly, but by investigating complex math ability, we cannot conclude whether the inaccuracy lies in the LLM's higher order mathematical reasoning, or it's understanding of basic operations. Thus, to study why LLMs do not perform well with mathematical tasks, we fed LLMs purely arithmetic queries to separate LLMs' semantic reasoning from its math capability, and its ability to write proofs from its basic ability to wield foundational mathematical techniques.

The structure of the prompts followed the few-shot prompting model where two example problems, "100 + 200 = 300`\n`520 + 890 = 1410`\n`", preceded an addition problem such as "478 + 23 = ". We chose to utilize the few-shot prompt model to condition Gemma towards a specific format.

C Extensions

As the presence of artificial intelligence continues to grow, the number of LLMs in exisence has followed suit. Above, we see the addition accuracy results of other popular LLMs. Architecture varies widely between models, but we hope to utilize our experiments to investigate the mechanisms of other LLMs and contribute to improving the accuracy and transparency of the inner workings of these black boxes.

D Additional Results

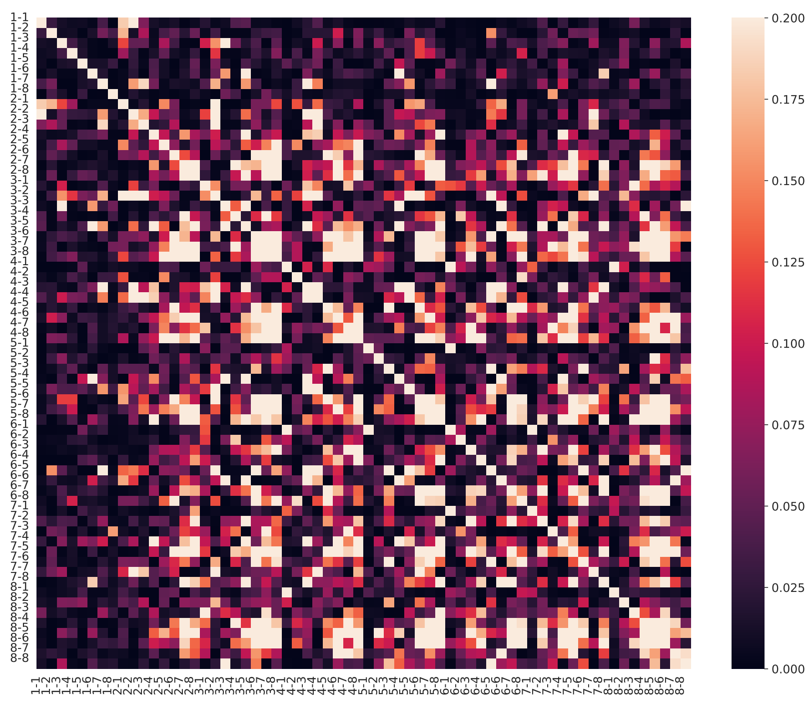

Figure 4: For a fixed model, `Gemma-2-2b`, this illustrates the Pairwise-Pearson correlations between the attention head contributions found through activation patching. Each cell intuitively captures the pairwise subcircuit similarity between subtasks (see the x, y-axes labels).

E Circuits

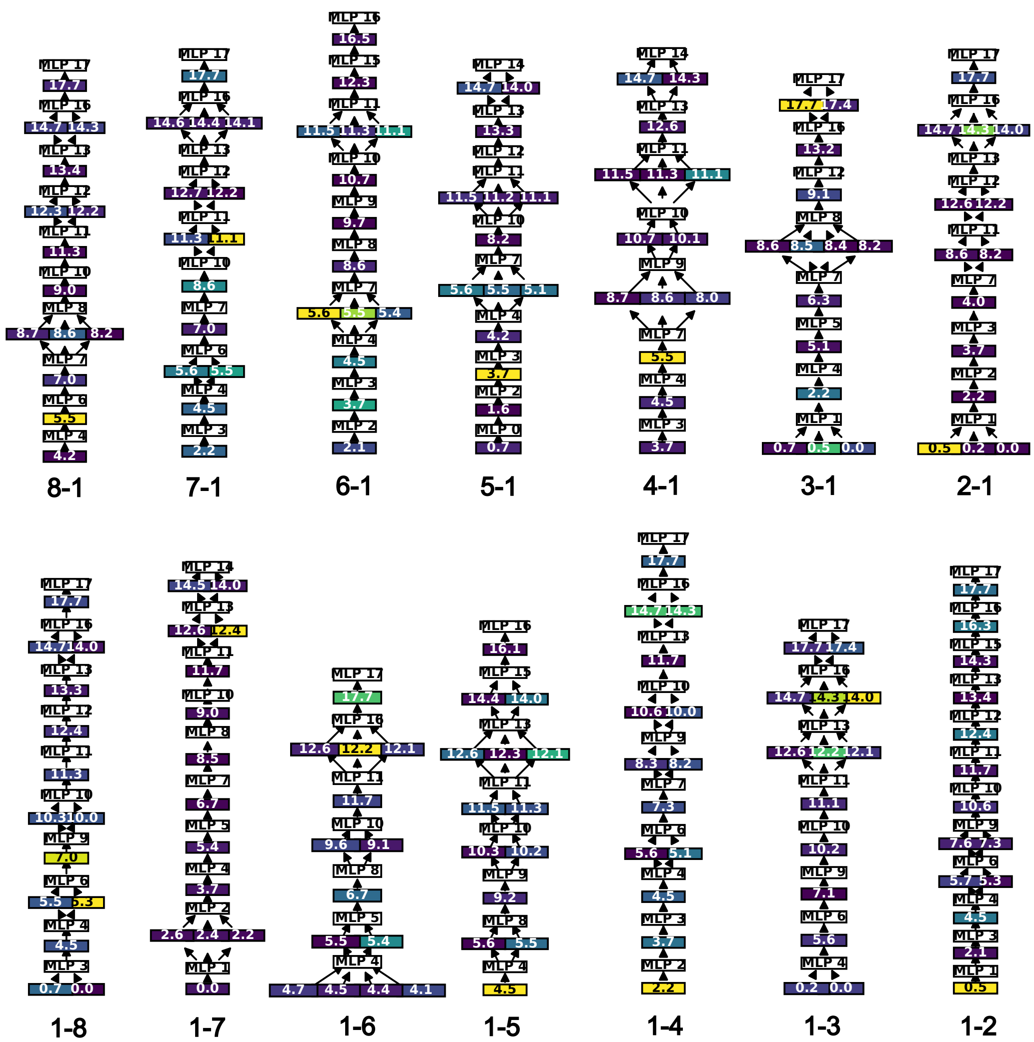

Figure 5: Circuits found for subtasks on the boundary. The attention heads shown are the top 10% of the most influential heads with respect to our patching metric. Since we do not patch any of the MLP layers, some of them are simply omitted from the graphs for brevity.

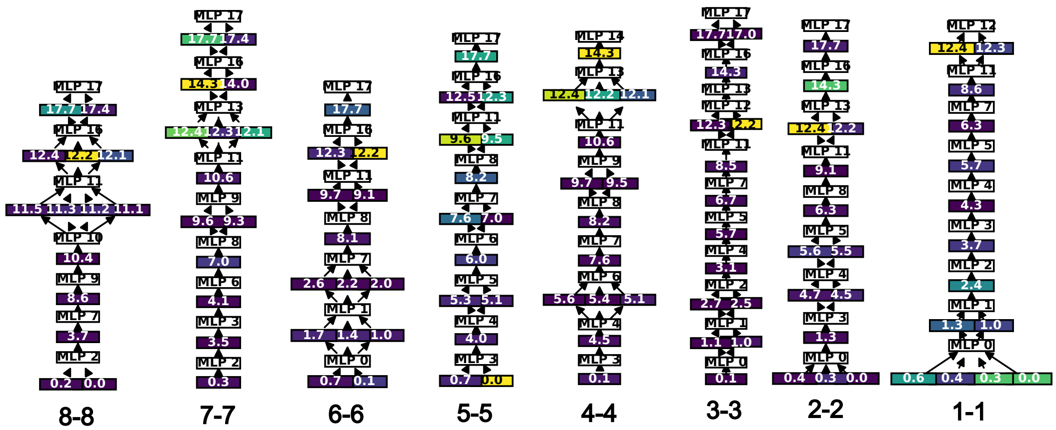

Figure 6: Circuits for the subtasks that are considered to be symmetric. The attention heads are the top 10% of the most influential heads with respect to our patching metric. Between any two attention heads from different layers, if there are not other influential attention heads between them we omit showing all of the MLPs between them for brevity sake. However, we do not patch or remove and of the MLPs.