NEURAL NETWORKS LEARN REPRESENTATION THEORY: REVERSE ENGINEERING HOW NETWORKS PERFORM GROUP OPERATIONS

Bilal Chughtai [*, †] **Lawrence Chan**[‡] **Neel Nanda**[†]

ABSTRACT

We present a novel algorithm by which neural networks may implement composition for any finite group via mathematical representation theory, through learning several irreducible representations of the group and converting group composition to matrix multiplication. We show small networks consistently learn this algorithm when trained on composition of group elements by reverse engineering model logits and weights, and confirm our understanding using ablations. We use this as an algorithmic test bed for the hypothesis of universality in mechanistic interpretability – that different models learn similar features and circuits when trained on similar tasks. By studying networks trained on various groups and architectures, we find mixed evidence for universality: using our algorithm, we can completely characterize the family of circuits and features that networks learn on this task, but for a given network the precise circuits learned – as well as the order they develop – are arbitrary.

1 INTRODUCTION

In this work, we study how networks implement composition of elements in arbitrary groups. This defines a large family of related tasks, forming an algorithmic test bed for investigating the universality hypothesis of mechanistic interpretability – that different networks learn similar features and algorithms. We begin by exhibiting a general algorithm by which networks can compute compositions of elements in an arbitrary finite group, using concepts from the mathematical field of representation[1] and character theory. We do this by building upon the work of Nanda et al. (2023), that reverse-engineered networks trained to grok modular addition and found the networks used a Fourier transform and trigonometry (trig) identity based algorithm to compute logits. We show that this ad-hoc, trig identity-based algorithm is a special case of our algorithm and that distinct Fourier modes correspond directly to *irreducible representations* of the cyclic group. Our algorithm and its implementation in network components is described in Figure 1.

Representation theory bridges linear algebra and group theory, and studies how group elements can be thought of as linear transformations. It is of universal importance to science; while groups encode the symmetries of physical systems, representations prescribe the set of possible actions of these symmetries on physical vector spaces. For instance, the representation theory of the particular Lie groups encoding symmetry transformations of spacetime determines the particles predicted by the standard model, which we observe in the universe.

At a high level, our algorithm embeds group elements as such matrices, uses its ReLU activations to perform matrix multiplication, and uses the unembed to convert back to group elements. We prove correctness of our algorithm using results from representation theory in Section 3. We verify networks implement this algorithm via four lines of evidence in Section 4. We then understand the progress of development of our algorithm over training via *progress measures*. Finally, we use our mechanistic understanding of our models to investigate the universality hypothesis in Section 5. We break universality down into strong and weak forms, finding evidence for weak universality, but against stronger forms.

[*]Corresponding author, please direct correspondence to: `brchughtaii@gmail.com`

[†]Independent.

[‡]University of California, Berkeley.

[1]We note our use of the word 'representation' is distinct to the usual use of the word representation in the ML literature.

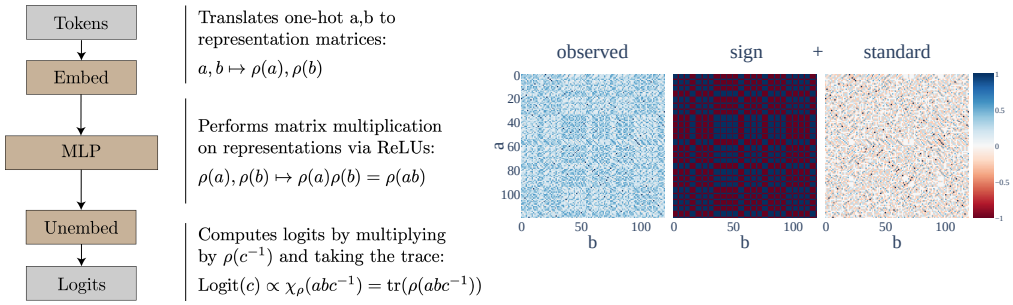

Figure 1: (**Left**) The algorithm implemented by a one hidden layer MLP for arbitrary group composition. Given two input group elements a and b, the model learns representation matrices $\rho(a)$ and $\rho(b)$ in its embeddings. Using the ReLU activations in its MLP layer, it then multiplies these matrices, computing $\rho(a)\rho(b) = \rho(ab)$. Finally, it 'reads off' the logits for each output group element c by computing *characters* – the matrix trace $\mathrm{tr}\,\rho(abc^{-1})$, denoted $\chi_\rho(abc^{-1})$, which is maximized when $c = ab$. (**Right**) The observed 0th logit over all pairs of inputs a (y-axis) and b (x-axis). The GCR algorithm's logit predictions χ_{sign} and $\chi_{standard}$ in the key representations. The observed logit appears to be a linear combination of the characters in the key representations. Note that all logits here have been normalized to range [-1, 1].

2 Setup and Background

2.1 Task Description

We train models to perform group composition on finite groups G of order $|G| = n$. The input to the model is an ordered pair (a, b) with $a, b \in G$ and we train to predict the group element $c = ab$. In our mainline experiment, we use an architecture consisting of left and right embeddings and [2], a one hidden layer MLP, and unembedding W_U. This architecture is presented in Figure 1 and elaborated upon in Appendix G. We note that the task presented in Nanda et al. (2023) is a special case of our task, as addition mod 113 is equivalent to composition for $G = C_{113}$, the cyclic group of 113 elements. We train our models in a similar manner to Nanda et al. (2023), details may be found in Appendix G.

2.2 Mathematical Representation Theory

A (real) **representation** is a homomorphism, i.e. a map preserving the group structure, $\rho : G \to GL(\mathbb{R}^d)$ from the group G to a d-dimensional general linear group, the set of real invertible square matrices of dimension d. Representations are in general *reducible*, in a manner we make precise in the Appendix. For each group G, there exist a finite set of fundamental **irreducible representations**. The **character** of a representation is the trace of the representation $\chi_\rho : G \to \mathbb{R}$ given by $\chi_\rho(g) = \mathrm{tr}(\rho(g))$. A key fact our algorithm depends on is that character's are maximal when $\rho(g) = I$, the identity matrix (Theorem H.8). In particular, the character of the identity element, $\chi_\rho(e)$, is maximal. We introduce all relevant results from representation theory in Appendix H. A simple example of these important definitions is presented in Appendix H.1

3 An Algorithm for Group Composition

We now present an algorithm, which we call group composition via representations (GCR), on an arbitrary group G equipped with a representation ρ of dimension d. The algorithm and it's map onto network components are described in Figure 1. We are not aware of this algorithm existing in any prior literature.

(1) Map inputs a and b to real $d \times d$ matrices $\rho(a)$, $\rho(b)$.
(2) Compute the matrix product $\rho(a)\rho(b) = \rho(ab)$.
(3) For each output logit c, compute the characters $\mathrm{tr}(\rho(ab)\rho(c^{-1})) = \mathrm{tr}(\rho(abc^{-1})) = \chi_\rho(abc^{-1})$.

Crucially, Theorem H.8 implies $ab \in \mathrm{argmax}_c\, \chi_\rho(abc^{-1})$, so that logits are maximised on $c^* = ab$, where $abc^{-1} = e$. If ρ is *faithful* (see Definition H.6), this argmax is unique.

[2]We do not tie the left and right embeddings as we study non abelian groups.

In our networks, we find the terms $\rho(a)$ and $\rho(b)$ in the embeddings and $\rho(ab)$ in MLP activations. Note, as $\rho(ab)$ is present in the final hidden layer activations and the unembed W_U learns $\rho(c^{-1})$ in weights, the map to logits is entirely linear:

$$\rho(ab) \to \operatorname{tr} \rho(ab)\rho(c^{-1}) = \sum_{ij} \left(\rho(ab) \odot \rho(c^{-1})^T\right)_{ij} \tag{1}$$

where $\odot$ denotes the element-wise product of matrices.

Empirically, we find networks often learn multiple representations, so that the observed logits are a linear combination of characters from multiple representations. Since any representation may be decomposed into a finite set of irreducible representations (Theorems H.4 and H.5), we may restrict our attention to irreducible representations. From now on, each representation may be assumed to be irreducible, and we will drop the word. Since each representation has $\chi_\rho(abc^{-1})$ maximized on the correct answers, using multiple representations gives constructive interference at $c^* = ab$, giving c^* a large logit.

Example. Our GCR algorithm is a generalization of the seemingly ad-hoc algorithm presented in Nanda et al. (2023) for modular addition, which in our framing is composition on the cyclic group of 113 elements, C_{113}. Each element of our algorithm maps onto their Fourier multiplication algorithm, via representations $\rho = \mathbf{2_k}$ (see Appendix H.2.1) corresponding to frequency $\omega_k = \frac{2\pi k}{n}$.

Nanda et al. (2023) found embeddings learn the terms $\cos(\omega_k a)$, $\sin(\omega_k a)$, $\cos(\omega_k b)$ and $\sin(\omega_k b)$, precisely the matrix elements of $\rho(a)$ and $\rho(b)$. The terms $\cos(\omega_k(a+b))$ and $\sin(\omega_k(a+b))$ found in the MLP neurons correspond directly to the matrix elements of $\rho(ab)$. Finally we find by direction calculation, or by using the group homomorphism property of representations, that the characters:

$$\chi(abc^{-1}) = \operatorname{tr}\left(\rho(abc^{-1})\right) = \operatorname{tr}\begin{pmatrix} \cos(\omega_k(a+b-c)) & -\sin(\omega_k(a+b-c)) \\ \sin(\omega_k(a+b-c)) & \cos(\omega_k(a+b-c)) \end{pmatrix} = 2\cos(\omega_k(a+b-c))$$

are precisely the form of logits found.

4 Reverse Engineering Permutation Group Composition in a One Layer ReLU MLP

We follow the approach of Nanda et al. (2023) in reverse engineering a single mainline model and then showing our results are robust later in Section 5. We produce several lines of mechanistic evidence that the GCR algorithm is being employed, mostly mirroring those in Nanda et al. (2023).

In our mainline experiment, we train the MLP architecture described in Section 2.1 on the permutation (or symmetric) group of 5 elements, S_5, of order $|G| = n = 120$. Note that unlike C_{113} studied by Nanda et al. (2023), S_5 is not abelian, so the composition is non-commutative. We present a detailed analysis of this case as symmetric groups are in some sense the most fundamental group, as every group is isomorphic to a subgroup of a symmetric group (Cayley's Theorem H.1). So, understanding composition on the symmetric group implies understanding, in theory, of composition on any group. The (non trivial) irreducible representations of S_5 are named sign, standard, standard_sign, 5d_a, 5d_b, and 6d, are of dimensions $d = \{1, 4, 4, 5, 5, 6\}$, and are listed in Appendix H.2.3. We summarize our findings here, going into further detail in Appendix A.

The GCR algorithm predicts that **logits** are sums of characters. This is a strong claim, which we directly verify in a black-box manner – we need not peer directly into network internals to check this. We do so by comparing the model's logits $l(a, b, c)$ on all input pairs (a, b) and outputs c with the algorithms character predictions $\chi_\rho(abc^{-1})$ for each representation ρ. Our model has logit similarity 0.509 with χ_{sign} and 0.767 with $\chi_{standard}$, and zero with all other representation characters (also see Figure 1). Theorem H.9 implies these character vectors are orthogonal, so we may approximate the logits with these two directions. Doing so explains 84.8% of the variance of logits. From now on we call these two representations the *key representations*.

The remainder of our approaches are white-box and involve direct access to internal model weights and activations. First, we inspect the **mechanisms** implemented in model weights. We find the embeddings and unembeddings to be completely memorized look up tables, converting inputs and outputs to the relevant representation elements $\rho(a)$ and $\rho(b)$ in the key representations (Table 1). As the number of representations learned is low, the embedding and unembedding matrices are low rank.

Table 1: Percentage of embedding matrices and MLP neurons explained by subspaces corresponding to representations in the GCR algorithm. We see the same two key representations explain almost all of the variance of each embedding matrix, and the non-key representations explain almost none. The MLP neurons calculate $\rho(ab)$ in the key representations, as well as other unimportant terms, as predicted by the algorithm.

	W_a	W_b	W_U	MLP $\rho(ab)$
SIGN	6.95%	6.95%	9.58%	2.51%
STANDARD	93.0%	93.0%	84.5%	10.5%
RESIDUAL	0.00%	0.00%	5.96%	87.0%

We then find **MLP neurons** calculate $\rho(ab)$ (Table 1), and are able to explicitly extract these representation matrices. We explain what the residual MLP neurons represent in Appendix A, and find they mostly lie in the null space of W_U so are not important. Additionally, MLP neurons cluster into distinct representations, and we can read off the linear map from neurons to logits as being precisely the final step of the GCR algorithm (Figure 7)

We use **ablations** to confirm our interpretation is faithful. We ablate components predicted by our algorithm to verify performance is hampered, and ablate components predicted to be noise, leaving only our algorithm, and show performance is maintained, or even sometimes improved.

We then introduce **progress measures** (Barak et al., 2023; Nanda et al., 2023) to understand the grokking behaviour of the network, finding that the network interpolates between a 'memorized' and 'generalized' solution in a manner than may be understood.

5 UNIVERSALITY

In this section, we investigate to what extent the universality hypothesis (Olah et al., 2020; Li et al., 2016) holds on our collection of group composition tasks. Here, 'features' correspond to irreducible representations of group elements[3] and 'algorithms' or 'circuits' correspond to precisely how networks manipulate these with their weights.

We interpret models of MLP and Transformer architectures (Appendix G) trained on group composition for seven groups: C_{113}, C_{118}, D_{59}, D_{61}, S_5, S_6 and A_5, each on four seeds. In Appendix B, we find evidence for *weak universality*: our models are *all* characterized by a family of circuits corresponding to our GCR algorithm across all possible group representations for a given group. We may faithfully track the progress of the computation described in Figure 1 through each component of each model. We however find evidence against *strong universality*: our models learn different representations: they do not predictably learn simpler representations over more complex representations and also learn different numbers of representations. We do however find lower dimensional representations are generally (but not always) learned first. This implies that specific features and circuits will differ across models.

6 CONCLUSION AND DISCUSSION

In this work, we use mechanistic interpretability to show that small neural networks perform group composition via an interpretable, representation theory–based algorithm, across several groups and architectures. We then define progress measures to study how the internals of networks develop over the course of training. We use this mechanistic understanding of models to study the universality hypothesis – that networks trained on similar tasks learn analogous features and algorithms. We find evidence for weak but not strong forms of universality: while all studied networks use a variant of the GCR algorithm, different networks (with the same architecture) may learn different sets of representations, and even networks that use the same representations may learn them in different orders. This suggests that reverse engineering particular behaviors in single networks is insufficient for fully understanding that behavior in general. That being said, even if strong universality fails in general, there is still promise that a 'periodic table' of universal features, akin to the representations in our group theoretic task, may exist in general for real tasks. We discuss some areas of future work in Appendix D.

[3]Defining a 'feature' in a satisfying way is surprisingly hard. Nanda (2022) discusses some of the commonly used definitions.

AUTHOR CONTRIBUTIONS

Bilal Chughtai was the primary research contributor and lead the project. He wrote the code, ran all experiments, reverse engineered the weights of the network trained on composition on S_5 in Section 4, and used this to automate the process of reverse engineering many more models in Section 5. He also wrote the paper.

Lawrence Chan provided significant help clarifying, framing and distilling the results, and with editing the final manuscript.

Neel Nanda supervised and mentored the entire project. He first proposed the GCR algorithm, and aided in editing the final manuscript.

ACKNOWLEDGMENTS

We are grateful to Joe Benton, Joseph Bloom, Ben Edelman, Jeremy Gillen, Stefan Heimersheim, Adam Jermyn, Cassidy Laidlaw, Eric Michaud and Martin Wattenberg for providing generous and valuable feedback on our manuscript. Over the course of the project, our thinking and exposition was also greatly clarified through correspondence with Spencer Becker-Kahn, Paul Colognese, Alan Cooney and Jacob Merizian.

We would additionally like to thank Joe Benton and Sam Marks for a conversation at a party that sparked this project, and to Sam for seeing the connection between representation theory and networks performing composition on S_5.

BC would like to thank the SERI MATS 2.1 program, particularly Joe Collman and Maris Sala, for providing an excellent research environment during the entire project. BC was also supported by SERI MATS for the duration of the project.

We trained our models using `PyTorch` (Paszke et al., 2019) and performed our data analysis using `NumPy` (Harris et al., 2020) and `Pandas` (McKinney, 2010). We made use of `SymPy` (Meurer et al., 2017) to handle permutation group operations, and `TransformerLens` (Nanda, 2023) to cache internal model activations for interpretability. Our figures were made using `Plotly` (Plotly Technologies Inc., 2015).

REFERENCES

J. L. Alperin and Rowen B. Bell. *Groups and Representations*, volume 162 of *Graduate Texts in Mathematics*. Springer, New York, NY, 1995. ISBN 978-0-387-94526-2 978-1-4612-0799-3. doi: 10.1007/978-1-4612-0799-3.

Yamini Bansal, Preetum Nakkiran, and Boaz Barak. Revisiting Model Stitching to Compare Neural Representations, June 2021.

Boaz Barak, Benjamin L. Edelman, Surbhi Goel, Sham Kakade, Eran Malach, and Cyril Zhang. Hidden Progress in Deep Learning: SGD Learns Parities Near the Computational Limit, January 2023.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners, July 2020.

Nick Cammarata, Shan Carter, Gabriel Goh, Chris Olah, Michael Petrov, Ludwig Schubert, Chelsea Voss, Ben Egan, and Swee Kiat Lim. Thread: Circuits. *Distill*, 5(3):e24, March 2020. ISSN 2476-0757. doi: 10.23915/distill.00024.

Xander Davies, Lauro Langosco, and David Krueger. Unifying Grokking and Double Descent. In *NeurIPS ML Safety Workshop*, December 2022.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep

Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits, 2021. https://transformer-circuits.pub/2021/framework/index.html.

Jonathan Frankle and Michael Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks, March 2019.

Deep Ganguli, Danny Hernandez, Liane Lovitt, Nova DasSarma, Tom Henighan, Andy Jones, Nicholas Joseph, Jackson Kernion, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Nelson Elhage, Sheer El Showk, Stanislav Fort, Zac Hatfield-Dodds, Scott Johnston, Shauna Kravec, Neel Nanda, Kamal Ndousse, Catherine Olsson, Daniela Amodei, Dario Amodei, Tom Brown, Jared Kaplan, Sam McCandlish, Chris Olah, and Jack Clark. Predictability and Surprise in Large Generative Models. In *2022 ACM Conference on Fairness, Accountability, and Transparency*, pp. 1747–1764, June 2022. doi: 10.1145/3531146.3533229.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2649-2.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of Neural Network Representations Revisited, July 2019.

Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John Hopcroft. Convergent Learning: Do different neural networks learn the same representations?, February 2016.

David Lindner, János Kramár, Matthew Rahtz, Thomas McGrath, and Vladimir Mikulik. Tracr: Compiled Transformers as a Laboratory for Interpretability, January 2023.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers Learn Shortcuts to Automata, October 2022a.

Ziming Liu, Ouail Kitouni, Niklas Nolte, Eric J. Michaud, Max Tegmark, and Mike Williams. Towards Understanding Grokking: An Effective Theory of Representation Learning, October 2022b.

Ziming Liu, Eric J. Michaud, and Max Tegmark. Omnigrok: Grokking Beyond Algorithmic Data, October 2022c.

Thomas McGrath, Andrei Kapishnikov, Nenad Tomašev, Adam Pearce, Demis Hassabis, Been Kim, Ulrich Paquet, and Vladimir Kramnik. Acquisition of Chess Knowledge in AlphaZero. *Proceedings of the National Academy of Sciences*, 119(47):e2206625119, November 2022. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.2206625119.

Wes McKinney. Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python in Science Conference*, pp. 56–61, 2010. doi: 10.25080/Majora-92bf1922-00a.

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. ISSN 2376-5992. doi: 10.7717/peerj-cs.103.

Neel Nanda. A Comprehensive Mechanistic Interpretability Explainer & Glossary. https://www.neelnanda.io/glossary, December 2022.

Neel Nanda. TransformerLens, January 2023.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability, January 2023.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. In Search of the Real Inductive Bias: On the Role of Implicit Regularization in Deep Learning, April 2015.

Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom In: An Introduction to Circuits. *Distill*, 5(3):e00024.001, March 2020. ISSN 2476-0757. doi: 10.23915/distill.00024.001.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context Learning and Induction Heads, September 2022.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library, December 2019.

Plotly Technologies Inc. Collaborative data science. https://plotly.com/, 2015.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets, January 2022.

Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, December 2017.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the Wild: A Circuit for Indirect Object Identification in GPT-2 small, November 2022.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent Abilities of Large Language Models, October 2022.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Thinking Like Transformers, July 2021.

Yi Zhang, Arturs Backurs, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, and Tal Wagner. Unveiling Transformers with LEGO: A synthetic reasoning task, July 2022.

A Reverse Engineering of Mainline Model

Here we give further evidence our mainline model trained on S_5 performs the GCR algorithm as described in Section 3, and give further details regarding our methods.

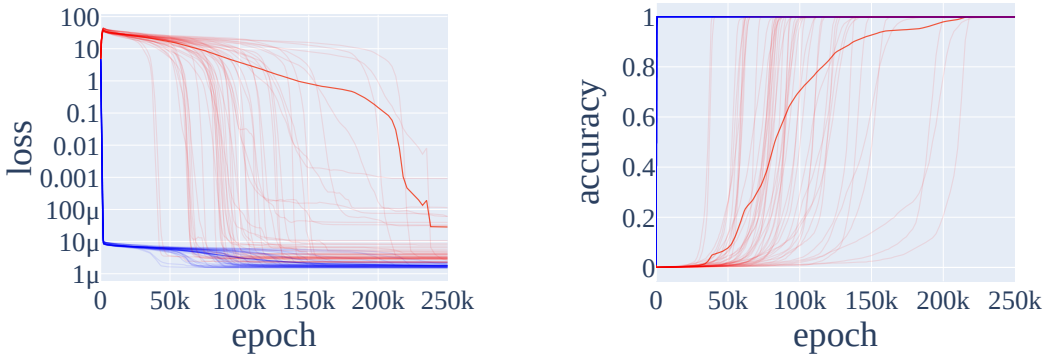

Figure 2: Mean train (blue) and test (red) loss (middle) and train and test accuracy (right) of 50 runs on different random seeds of our mainline configuration: an MLP trained on group composition on S_5, the permutation group of order 5. These models consistently exhibit grokking: they quickly overfit early in training, but then suddenly generalise much later. The bolded line denotes average loss/accuracy.

A.1 Logit Attribution

Logit similarity. We call the correlation between the logits $l(a, b, c)$ and representation characters $\chi_\rho(abc^{-1})$ the *logit similarity*. We call representations with logit similarity (see Appendix A.10) greater than 0.005 'key'.

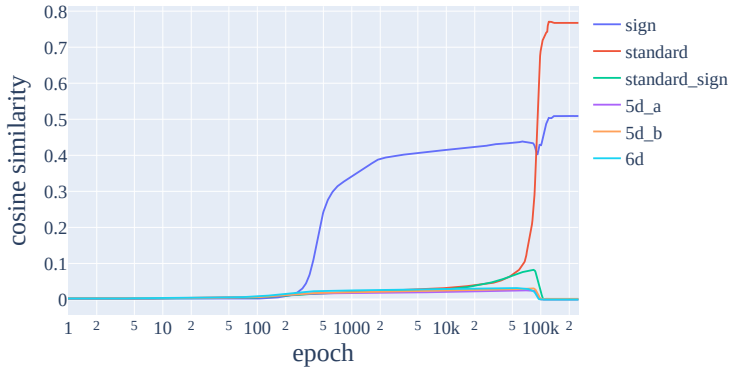

Figure 3: Evolution of logit similarity over training for each of the six non trivial representations. We see the sign representation is learned around epoch 250, and the standard around epoch 50k. None of the other representations contribute to logits via the GCR algorithm at the end of training. We therefore call the sign and standard representations 'key'.

Our model has logit similarity 0.509 with χ_{sign} and 0.767 with $\chi_{standard}$, and zero with all other representation characters. Theorem H.9 implies these character vectors are orthogonal, so we may approximate the logits with these two directions. Doing so explains 84.8% of the variance of logits. This is surprising – the 120 output logits are explained well by only two directions. As confirmation for the correctness of our algorithm, if we evaluate test loss only using this logit approximation, we see a *reduction* in loss by 70% relatively. If we ablate the remaining 15% of logits, loss does not change.

A.2 Embeddings and Unembeddings

Each representation is a set of n $d \times d$ matrices, which by flattening we can think of as a set of d^2 vectors of dimension n. We call the subspace of $\mathbb{R}^n$ spanned by these vectors *representation space*. Theorem H.10 implies these subspaces are orthogonal for distinct representations, and Theorem H.4

implies the direct sum of each of these subspaces over all representations is $\mathbb{R}^n$. Any embedding or unembedding of n group elements lies in $\mathbb{R}^{n \times h}$ for some h, so a natural operation is to project embeddings and unembeddings onto representation space over the n dimension. Details regarding how we do this, and our definitions of embedding matrices W_a, W_b and W_U may be found in Appendices G.1 and A.10.

Table 2: Percentage of embedding matrices explained by subspaces corresponding to representations. We see the same two key representations explain almost all of the variance of each embedding matrix, and the non-key representations explain almost none.

	W_a	W_b	W_U
SIGN	6.95%	6.95%	9.58%
STANDARD	93.0%	93.0%	84.5%
RESIDUAL	0.00%	0.00%	5.96%

We find evidence of representations in embeddings and unembeddings. We find that the embedding matrices and the unembed matrix are well approximated by a sparse set of representations (Table 2), and that the representations contained in all three are the same. This is surprising: each embedding and unembedding can potentially be of rank 120, but is only of rank $16+1$, corresponding precisely to the two key representations. Qualitatively, the progress of representation learning is similar across all three embedding and unembedding matrices, with each representation being learned suddenly at roughly the same time, see Figure 4.

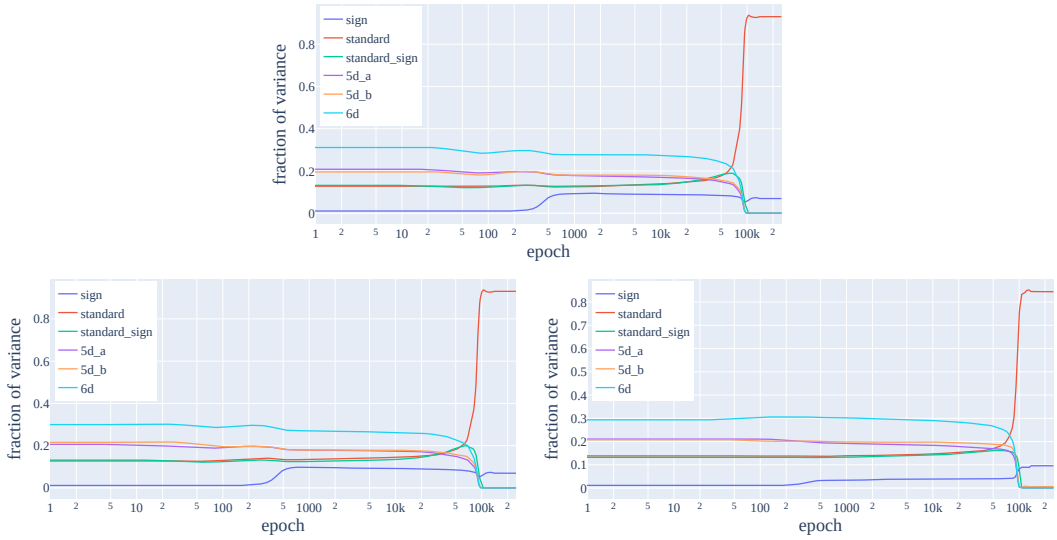

Figure 4: Evolution of the fraction of the left embedding (top left), right embedding (top right), and unembedding (bottom) explained by $\rho(a)$, $\rho(b)$ and $\rho(c^{-1})$ respectively. Representations are learned suddenly and at approximately the same time across all the embeddings, evidence that they are learned as part of the GCR algorithm. As the representation spaces form an orthogonal decomposition of $\mathbb{R}^n$, the terms will always add up to 1, we draw the reader's attention to the sparsity over embeddings. At initialization, each representation explains $d^2/|G|$ of the embedding due to randomness.

A.3 MLP NEURONS

MLP neurons calculate $\rho(ab)$. From the embeddings, neurons have inputs $\rho(a)$ and $\rho(b)$, and use their non-linearity to calculate $\rho(ab)$. We make this calculation explicit in the 1d case in Appendix A.7. To demonstrate this, we follow the approach taken with embeddings. We define for each representation a *hidden representation subspace* of rank d^2 of $\mathbb{R}^{n^2}$, and consider the projection of the hidden layer onto these subspaces.

Table 3: Percentage of the variance of MLP neurons explained by subspaces corresponding to representations of group elements a, b and ab. Almost all of the variance of neurons within each key representation cluster is explained by subspaces corresponding to the representation, and all neurons are in a single cluster.

CLUSTER	$\rho(a)$	$\rho(b)$	$\rho(ab)$	RESIDUAL
SIGN	33.3%	33.3%	33.3%	0.00%
STANDARD	39.6%	37.1%	11.3%	12.1%

Neurons cluster by representation. Our neurons cluster into disjoint categories, corresponding to key representations. This clustering is identical on neuron inputs and outputs. 7 neurons are 'sign neurons': these neurons completely represent $\rho_{sign}(a)$ in the left embedding and $\rho_{sign}(b)$ in the right embedding. On post-activation outputs, they represent some linear combination of $\rho_{sign}(a)$, $\rho_{sign}(b)$, and $\rho_{sign}(ab)$, but not any other representation. 119 neurons are correspondingly 'standard neurons'. The final 2 neurons are always off.

In Table 3 we find 88.0% of the variance of standard neurons can be explained by the directions corresponding to $\rho(a)$, $\rho(b)$ and $\rho(ab)$. For sign neurons, this fraction of variance of neurons explained is 99.9%. We validate by ablation the residual 12.0% of standard neurons does not affect performance. We hypothesize this term is a side product of the network performing multiplication with a single ReLU, and discuss this multiplication step further in Appendix A.7. Evolution of percentage of MLP activations explained by each representation is presented in Figure 5.

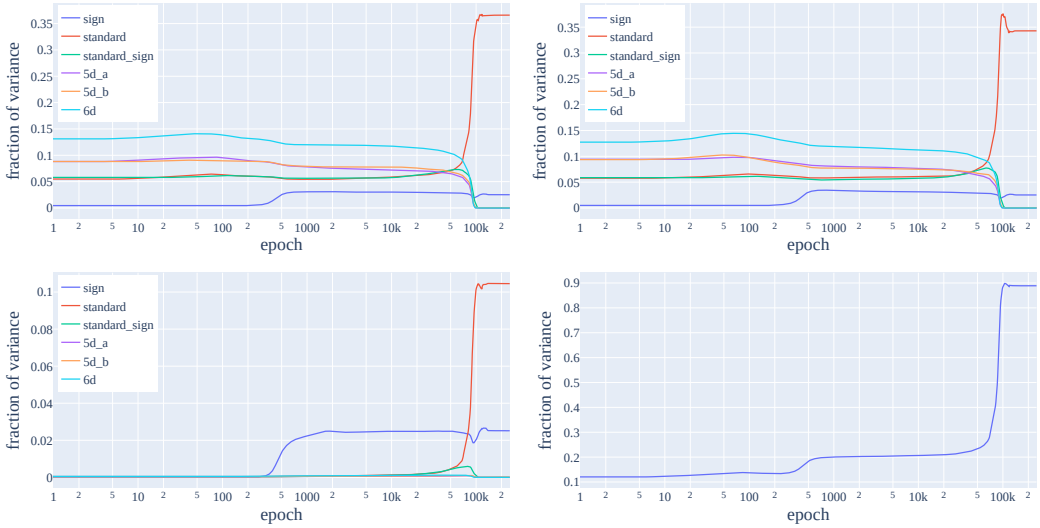

Figure 5: Evolution of the fraction of the MLP neurons explained by $\rho(a)$ (top left), $\rho(b)$ (top right), $\rho(ab)$ (bottom left), and the sum of all three over all representations (bottom right). These track the same timing as representation learning in the embeddings and unembeddings, further evidence for our algorithm. Note that in order to perform step 2 in the GCR algorithm, $\rho(ab)$ must be calculated. If a representation has $\rho(a)$ and $\rho(b)$ represented but *not* $\rho(ab)$ then the representation has not been learned.

Only the $\rho(ab)$ component of MLP neurons is important. The GCR algorithm doesn't make use of $\rho(a)$ or $\rho(b)$ directly to compute $\chi_\rho(abc^{-1})$. We confirm the model too only makes use of $\rho(ab)$ type terms by ablating directions corresponding to $\rho(a)$ and $\rho(b)$ or otherwise in MLP activations and verifying loss doesn't change.

On the other hand, ablating directions corresponding to $\rho(ab)$ in the key representations severely damages loss. Baseline loss is 2.38×10^{-6}. Ablating $\rho_{standard}(ab)$ increases loss to 7.55, while ablating $\rho(ab)_{sign}$ increases loss to 0.0009.

We may explicitly recover representation matrices from hidden activations. By changing basis (via Figure 6) on the hidden representation subspace corresponding to $\rho(ab)$, we may recover the matrices $\rho(ab)$. The learned sign representation matrices agree with $\rho_{sign}(ab)$ completely, and the learned standard representation matrices agree with $\rho_{standard}(ab)$ with MSE loss $< 10^{-8}$. We cannot recover representation matrices for representations not learned.

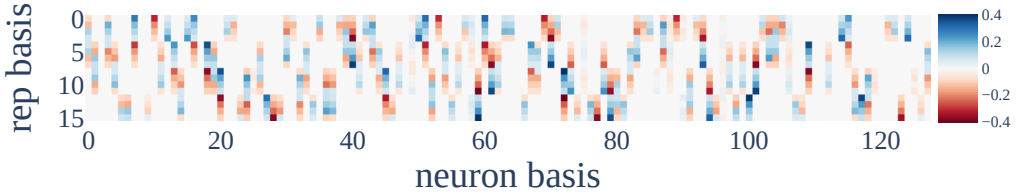

Figure 6: Change of basis matrix from projected MLP space standard representation space. Note some neurons correspond to blocks of 4 cells in the representation basis – these correspond to standard representation matrix rows. Neurons in other clusters can be explicitly seen as being off in this change of basis matrix.

A.4 Logit Computation

Maps to the logits are localised by representation. The unembedding map W_U restricts to each key representation neuron cluster. This restricted map, following a similar approach to Section A.2, has components only in the corresponding output representation subspace. Defining W_ρ as the map from ρ-neurons to logits, we find W_{sign} has 99.9% variance explained by output sign representation space, and $W_{standard}$ has 93.4% explained by output standard representation space.

The linear map in representation basis. As noted in Section 3, the final step of the GCR algorithm may be implemented in a single linear operation (Equation 1). Given $\rho(ab)$ is present in MLP neurons, the unembedding need simply learn the inverse representation matrices $\rho(c^{-1})$. We verify the network implements this step as predicted by our algorithm in Figure 7.

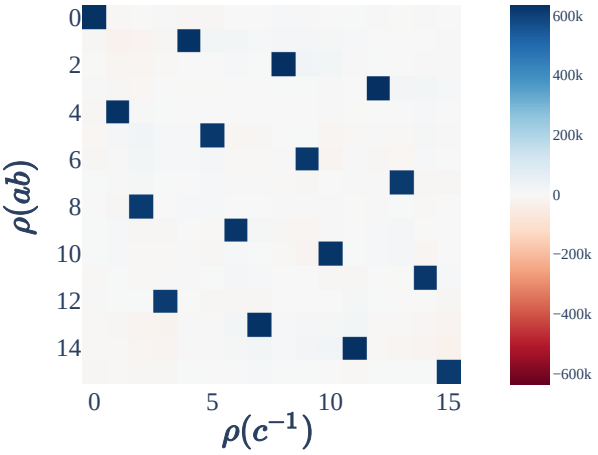

Figure 7: The map from the subspace corresponding to $\rho_{standard}(ab)$ in the MLP neurons to logits. We obtain this by changing basis of W_U on both sides, to align with $\rho(ab)$ representation space on the left, and $\rho(c^{-1})$ on the right. This matrix implements step 3 in the GCR algorithm, mapping $\rho(ab)$ to $\chi_\rho(abc^{-1}) = tr(\rho(ab)\rho(c^{-1}))$. The sparse and uniform matrix shown corresponds precisely to the trace calculation between two 4×4 matrices as in Equation 1.

A.5 Correctness Checks: Ablations

In previous sections, we showed various components of the model were well approximated by intermediate terms of the proposed GCR algorithm. To verify these approximations are faithful, we perform two types of additional ablations. We **exclude** components in the algorithm and verify loss increases, and we **restrict** to these same components and demonstrate loss remains the same or decreases.

MLP neurons. In Section A.3, we identified sets of neurons that could be manipulated to recover representation matrix elements $\rho(ab)$. If we replace these neurons with the corresponding representation matrix elements directly, we find loss decreases by 70% (to 7.00×10^{-7}).

Unembeddings. In Section A.4, we found W_U is well approximated by $16 + 1$ directions, corresponding to representation space on the two key representations. If we project MLP neurons to only these directions, ablating the 5.96% residual in W_U, we find loss decreases by 12%, while if we project to only this residual, loss increases to 4.80, random.

Logits. In Section A.1 we found observed logits were well approximated by the GCR algorithm in the key representations. We find ablating our algorithm's predictions in the key representations damages loss, to 0.0006 by excluding the sign representation, to 7.23 excluding the standard representation, and to 7.60 excluding both, significantly worse than random. Ablating other directions improves performance.

A.6 Understanding Training Dynamics using Progress Measures

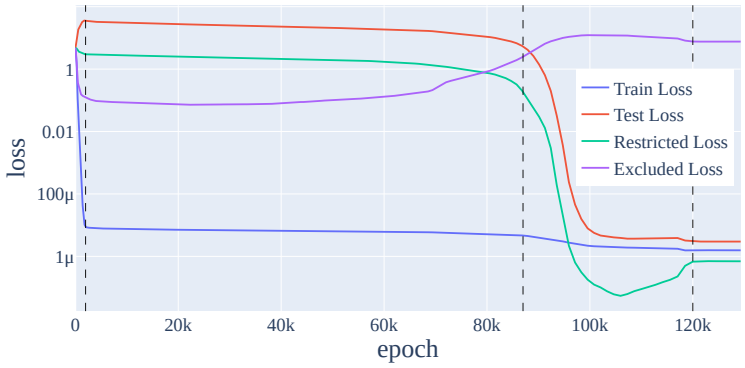

Figure 8: Evolution of the two progress measures over training. The vertical lines delineate 3 phases of training: memorization, circuit formation, and cleanup (and a final stable phase). Excluded loss tracks the progress of the memorization circuit, and accordingly falls during the first phase, rising after during circuit formation and cleanup. Restricted loss tracks the progress of the generalized algorithm, and has started falling by the end of circuit formation. Note that grokking occurs during cleanup, only after restricted loss has started to fall.

A limitation of prior work on using *hidden progress measures* from mechanistic explanations as a methodology for understanding emergence (Nanda et al., 2023) is that the techniques developed may not generalize beyond one specific task. We demonstrate their results are robust by replicating them in our network trained on S_5.

We argue that the network implements two classes of circuit – first, 'memorizing' circuits, and later, 'generalizing' circuits. Both are valid solutions on the training distribution – To disentangle these, we define two progress measures. **Restricted loss** tracks only the performance of the generalizing circuit via our algorithm. **Excluded loss** is the opposite, tracking the performance of only the memorising circuit, and so is only evaluated on the training data. We find that on our mainline model, training splits into three partially overlapping phases – memorisation, circuit formation, and cleanup. During circuit formation, the network smoothly transitions from memorising to generalising. Since test performance requires a general solution and no memorisation, grokking occurs during cleanup. Further discussion may be found in Appendix A.6.

We give more full definitions of our progress measures below.

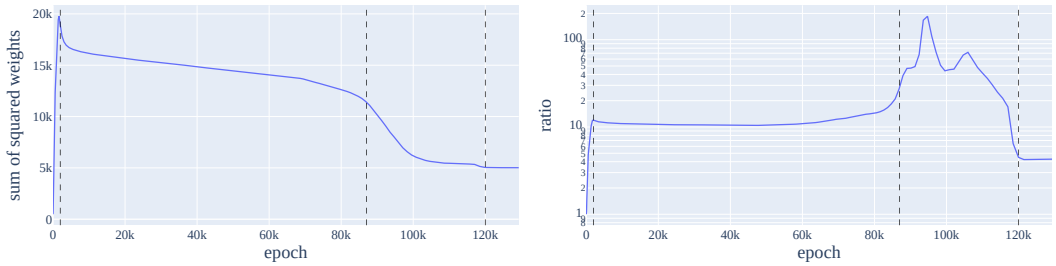

Figure 9: The sum of squared weights (left), and ratio of test loss and restricted loss (right). The sum of squared weights decreases smoothly during circuit formation and more sharply during cleanup, indicating both phases are linked to weight decay. (Right) Intuitively, restricted loss is us artificially cleaning up some the model (besides W_U), while test loss requires both circuit formation and cleanup. So a large discrepancy shows the rate of circuit formation outstrips the rate of cleanup during grokking.

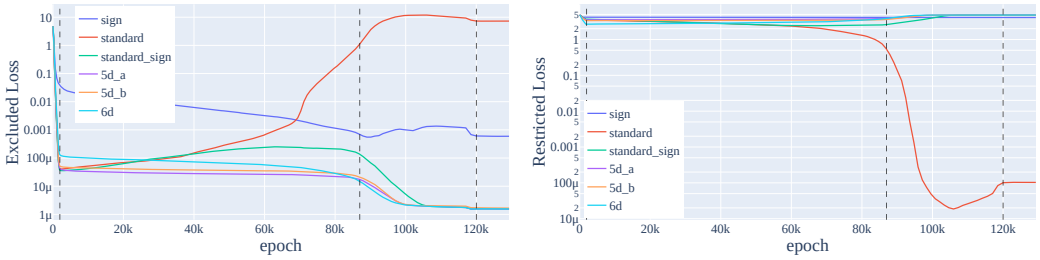

Figure 10: Excluded (left) and restricted loss (right), separated out by representation. As with the results of Section A.6, this shows the model interpolates between memorizing and generalizing. In the restricted loss plot, we see the sign representation is incapable of solving the task alone, but contributes several orders of loss improvement when coupled with the standard representation, as can be seen in excluded loss.

Restricted Loss. We restrict the MLP activations to the terms corresponding to $\rho(ab)$ in the key representations, a $16 + 1$ dimensional subspace of $\mathbb{R}^{128}$, and then map this restricted MLP layer to logits. By doing so, we isolate the performance of the generalising algorithm. This assumes that the memorising algorithm has no privileged subspace in the MLP layer.

Excluded Loss. The opposite of restricted loss. Instead of keeping the key representations, we remove only those representations from the MLP neurons, and see how this affects loss. Having removed the generalising solution, this isolates the performance of the memorising solution. This therefore makes sense to measure only on the *training* data, which we do.

The three phases of training we define are as follows, and can be seen in Figures 8, 9 and 10.

Memorization. (Epochs 0-2k) We first observe a decline of both excluded and train loss, with test and restricted loss both remaining high. In other words, the model memorizes the training data. The sum of squared weights peaks at the end of memorization, so weight decay does not prefer these memorized circuits. As test loss increases but restricted loss stays constant as no progress towards generalization is made, the ratio of test loss to restricted loss rises.

Circuit Formation. (Epochs 2.2k-87k) In this phase, excluded loss rises, sum of squared weights falls (Figure 9), restricted loss starts to fall, and train and test loss stay flat. This suggests that the models behavior on the train set transitions smoothly from the memorizing solution to the generalizing solution. The fall in the sum of squared weights suggests that circuit formation likely happens due to weight decay. Notably, the circuit is formed well before grokking.

Cleanup. (Epochs 87k-120k) In this phase, restricted loss continues to drop, test loss suddenly drops, sum of squared weights sharply drops, and the ratio of test to restricted loss is variable and

then sharply decreases (Figure 9). As the generalizing circuit both solves the task well and has lower weight at comparable performance as compared with memorization circuits on the training set, weight decay encourages the network to shed the memorized solution. Weight decay contributes an important inductive bias of our networks (Neyshabur et al., 2015). The slight rise in restricted loss at the very end of training is too a result of weight being traded off against performance – multiplying the entire circuit by a fixed constant $r > 1$ will reduce loss, though also requires more weight.

We sometimes find further phase changes. Figure 11 demonstrates two *phases of grokking* in a seperate run, caused by learning of different representations at distinct times.

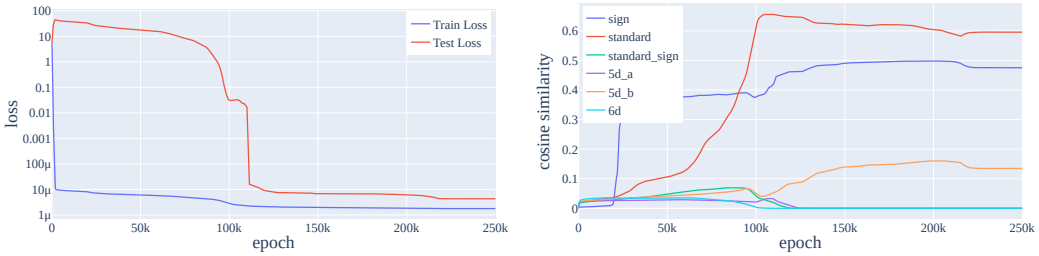

Figure 11: (Left) Train and test loss of a different MLP random seed trained on S_5. (Right) Logit similarity of this run over training. We see two phases of grokking. The model initially groks as the memorizing circuit is cleaned up in presence of the valid general standard circuit. Loss then plateaus as the 5d_b circuit is learned around epoch 100k, before the model groks again as cleanup continues.

A.7 Full Circuit Analysis: Sign Representation

In Nanda et al. (2023), the authors primarily analyse 2d representations via Fourier transforms, and we primarily analyse 4d standard representations in our mainline model. Treating sines and cosines as separate objects adds complexity, which we avoid by unifying them as matrix elements of the same representation. However, two dimensional features retain some redundancy over choice of basis, or equivalently, choice of rotation axis. So in general, some manipulation of activations and weights is necessary to interpret the model.

The sign representation on the other hand is a one dimensional representation of certain groups. This computational subgraph may be understood by directly inspecting activations and weights, without ever having to change basis. We demonstrate this simplicity on our mainline model.

MLP neuron activations are 'blocky'. We can identify interpretable activation patterns by inspection. Working backwards we identify embeddings directly learn $\pm sign(a)$ and $\pm sign(b)$.

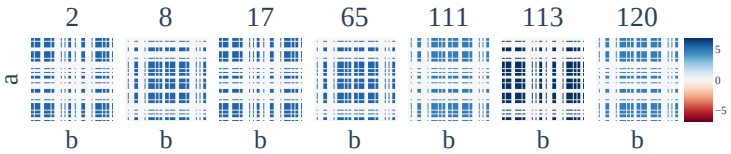

Figure 12: The seven 'sign neuron' activations over the whole distribution of inputs. Each activates uniformly on inputs, with form some multiple of $\mathbb{1}(sign(a) = \pm 1)\mathbb{1}(sign(b) = \pm 1)$, where $\pm$ are independent.

We then can write out, for x, y some positive constants and n_i neuron i:

$$\begin{aligned} n_2 &= xReLU\left(+sign(a) + sign(b)\right) \\ n_8 &= xReLU\left(-sign(a) - sign(b)\right) \\ n_{17} &= xReLU\left(+sign(a) + sign(b)\right) \\ n_{65} &= xReLU\left(-sign(a) - sign(b)\right) \\ n_{111} &= xReLU\left(+sign(a) - sign(b)\right) \\ n_{113} &= yReLU\left(-sign(a) + sign(b)\right) \\ n_{120} &= xReLU\left(+sign(a) - sign(b)\right) \end{aligned}$$

In general, interpreting the matrix multiplication operation is challenging, though in the one dimensional case it turns out to be simple. We see that the MLP performs multiplication of signs via ReLU and addition. For instance

$$n_2 + n_8 + n_{111} + n_{113} = 2x \times sign(a) \times sign(b)$$

This is essentially computing an XOR gate on the inputs, so is not multiplication of arbitrary inputs, which is why the network can implement this operation perfectly. Note that we need a minimum of four neurons to implement this operation in this manner [4]. Empirically, we found that the number of sign neurons was often four exactly. In this case, neuron 113 appears to be used in two such multiplication calculations.

We expect that higher dimensional matrix multiplication is implemented similarly – see further discussion in Appendix A.8

Map to logits. Calling neurons 2, 8, 17 and 65 positive, and neurons 111, 113, and 120 negative, we find that $W_U|_+ \sim +sign(c^{-1})$ and $W_U|_- \sim -sign(c^{-1})$, thus this circuit contributes positively to logits on correct signs and negatively to wrong signs, giving a contribution $\chi_{sign}(abc^{-1})$ to logits.

A.8 Implementing Multiplication via ReLUs

Here we briefly discuss how networks may implement multiplication in a single layer. Our GCR algorithm necessitates this in step 2, and we provide a simple example of this occurring in Appendix A.7.

Network's can multiply activations to some extent in one layer, though may not be able to do so perfectly, and also may put information into additional directions (as can be seen by the 12% residual of standard MLP neurons in Section A.3). Note in our context multiplication is not generic multiplication, but multiplication of elements in a fixed range; most of our representation matrices have entries $\{0, -1, 1\}$ on which multiplication can be implemented in a finite set of ReLU's with a bias as for instance

$$x \times y = ReLU(x + y - 1) + ReLU(-x - y - 1) - ReLU(x - y - 1) - ReLU(-x + y - 1)$$

Changing the network architecture may aid it's ability to perform multiplication. Changing activation function to x^2 for instance permits exact multiplication as

$$x \times y = \frac{1}{4}\left((x + y)^2 - (x - y)^2\right)$$

We hypothesize that the number of neurons in each representation cluster learned is linked to the number of such ReLU activations required to compute the matrix multiply, explaining why we have many more standard neurons than sign after accounting for higher dimensionality of representation. Of course networks won't implement elementwise multiplication but rather some efficient matrix algorithm, such as Strassens.

[4] If $x, y \in \{0, 1\}$ then x XOR $y = ReLU(x - y) + ReLU(y - x)$ is a solution in two neurons.

A.9 VISUALIZING THE EMBEDDINGS AND UNEMBEDDINGS.

Power et al. (2022) found it useful to use t-SNE to vizualize the unembedding in their networks trained on S_5. Here, we replicate their results, and show an additional meaningful representation in Figure 13. We did not find the unembedding W_U to cluster into subgroups as they did via t-SNE, but did via PCA. We did find clustering in embeddings into cosets of a subgroup of S_5, though found it to be a different subgroup. Over different run's, these subgroups were arbitrary, though given the stochastic nature of t-SNE it is hard to say whether this observation is meaningful.

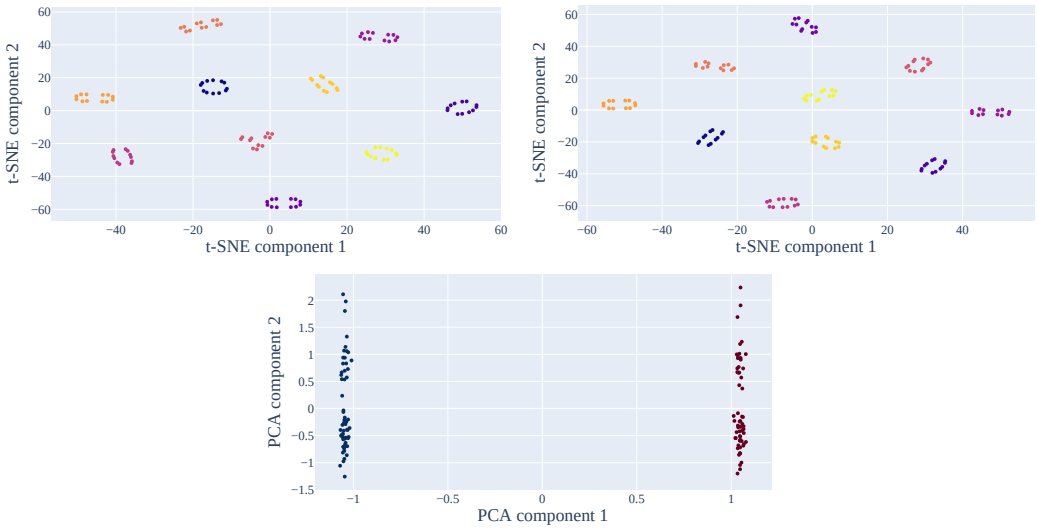

Figure 13: Left embedding (left) and right embedding (right) visualized in two dimensions via t-SNE. We see a large amount of structure. Clusters correspond to cosets of an order 12 subgroup of S_5. (Bottom) Visualization of the unembedding via PCA. The two clusters correspond to cosets of A_5, the alternating group i.e. the sign of group elements.

A.10 FURTHER REVERSE ENGINEERING DETAILS

Logit Similarity. Observed logits $l(a, b, c)$ are an n^3 dimensional tensor over all input pairs (a, b) and outputs c. The GCR algorithm's character predictions $\chi(abc^{-1})$ are also an n^3 tensor. We compute the correlation of these by flattening each tensor into a vector of dimension n^3, and computing the cosine similarity of these.

Representation Space. Each representation gives a set of n $d \times d$ matrices, one for each group element. We wish to investigate to what degree these are present in various model weights or activations. We can think of each representation as an $n \times d^2$ tensor of flattened matrices R. We call the n-dimensional space spanned by these d^2 columns *representation space*. In order to project onto this space, we apply QR decomposition to R, obtaining $\tilde{R}$.

Any embedding or unembedding can be thought of an an $n \times h$ tensor W. Then $\tilde{R}^T W$ is a $d^2 \times h$ matrix. By inspecting the h dimension of this matrix, we may understand neuron clustering, and by comparing the norm of it relative to the norm of the embedding or unembedding, we understand the percentage contribution of the subspace. An entirely analogous methodology is applied to understanding the MLP neurons via hidden representation spaces.

Centering. Neural network activations often contain large biases, even without the presence of explicit bias terms in the architecture. MLP neurons follow a ReLU activation, so necessarily have a mean positive activation. Accounting for this would artificially increase all 'fraction of variance explained' metrics. To avoid this, we remove this bias by subtracting the mean over the batch dimension before interpreting the MLP activations.

Similarly, since softmax is a function of relative logit difference, on each fixed input logits have some learned and unimportant bias. Accounting for this would artificially contribute to 'logit similarity' under the trivial representation, and artificially increase the fraction of logits explained metric. To avoid this, we remove this by subtracting the mean over output dimension.

B UNIVERSALITY RESULTS

Do models converge on the same solutions to a task, or are the algorithms implemented arbitrary and unpredictable? The *universality hypothesis* (Olah et al., 2020; Li et al., 2016) asserts that models learn similar features and circuits across different models when trained on similar tasks. This is an open question of significant importance to the field of *mechanistic interpretability*. The field focuses on reverse engineering state-of-the-art models by identifying circuits (Elhage et al., 2021; Olsson et al., 2022; Nanda et al., 2023; Wang et al., 2022), subgraphs of a network consisting of a set of tightly linked features and the weights between them.(Olah et al., 2020). Recently, the field of mechanistic interpretability has increasingly shifted towards finding small, toy models easier to interpret, and employing labor intensive approaches to reverse-engineering specific features and circuits in detail (Elhage et al., 2021; Wang et al., 2022; Nanda et al., 2023). If the universality hypothesis holds, then the insights and principles found by studying small models will transfer to state-of-the-art models that are used in practice. But if universality is false, then although we may learn some general principles from small models, we should shift focus to developing scalable, more automated interpretability techniques that can directly interpret models of genuine interest.

Here we provide further discussion on our results on universality from studying group composition on several groups and architectures.

All trained networks implement the GCR algorithm. We first argue for weak universality via universality of our algorithm and universality of a *family* of features and circuits involving them. Following the approach of Section 4, we understand each layer of our network as steps in the algorithm as presented in Section 3. (Table 4). Steps 1 and 3 – we analyze embedding and unembedding matrices, showing that their fraction of variance explained (FVE) by subspaces corresponding to the key representations is high. Each group has its own family of representations, and each model learns its own set of key representations (i.e. representations with non-zero logit similarity). Where applicable, our metrics track only these key representations of any given model. For Step 2, we show the MLP activations are well explained by the terms $\rho(a)$, $\rho(b)$, and importantly $\rho(ab)$ in the key representations. Finally, as evidence our algorithm is entirely responsible for performance, we show the final values of the progress measures of restricted and excluded loss.

Specific representations learned vary between random seeds. Each group has several representations that can be learned. Under strong universality, we would expect the representations learned to be consistent across random seeds when trained on the same group. In general, we do not find this to be true (Figure 14). When there are multiple valid solutions to a problem, the model somewhat arbitrarily chooses between them – even when the training data and architecture are identical.

It is not the case that networks learn simple representations over complex representations. If strong universality is true, we hypothesized networks would learn 'simple' representations over more complex ones, according to some sensible measure of complexity.

Table 4: Results from all groups on both MLP and Transformer architectures, averaged over 4 seeds. We find that that features for matrices in the key representations are learned consistently, and explain almost all of the variance of embeddings and unembeddings. We find that terms corresponding to $\rho(ab)$ are consistently present in the MLP neurons, as expected by our algorithm. Excluding and restricting to these terms in the key representations damages performance/does not affect performance respectively.

	MLP								**Transformer**						
	FVE					Loss			FVE				Loss		
Group	W_a	W_b	W_U	*MLP*	$\rho(ab)$	*Test*	*Exc.*	*Res.*	W_E	W_L	*MLP*	$\rho(ab)$	*Test*	*Exc.*	*Res.*
C_{113}	99.53%	99.39%	98.05%	90.25%	12.03%	1.63e-05	5.95	6.88e-03	95.18%	99.52%	92.12%	16.77%	2.67e-07	9.42	2.12e-02
C_{118}	99.75%	99.74%	98.43%	95.84%	13.26%	5.39e-06	8.72	3.60e-03	94.05%	99.64%	94.63%	17.11%	1.73e-07	15.93	2.55e-01
D_{59}	99.71%	99.73%	98.52%	87.68%	12.44%	6.34e-06	12.37	1.60e-06	98.58%	98.53%	85.01%	10.85%	3.20e-06	46.42	2.82e-05
D_{61}	99.26%	99.45%	98.26%	87.61%	12.48%	1.79e-05	12.00	1.69e-06	98.33%	97.40%	85.59%	11.11%	1.63e-02	41.64	9.60e-02
S_5	100.00%	99.99%	94.14%	88.91%	12.13%	1.02e-05	11.72	2.21e-07	99.84%	99.97%	85.28%	10.23%	1.43e-07	17.77	4.44e-09
S_6	99.65%	99.78%	93.67%	86.38%	8.98%	4.95e-05	12.17	2.66e-06	99.94%	99.93%	86.32%	9.35%	2.21e-06	291.67	1.05e-06
A_5	99.04%	99.31%	93.27%	86.69%	10.26%	1.94e-05	9.82	5.28e-07	97.53%	97.40%	83.56%	8.22%	4.88e-02	19.76	7.70e-04

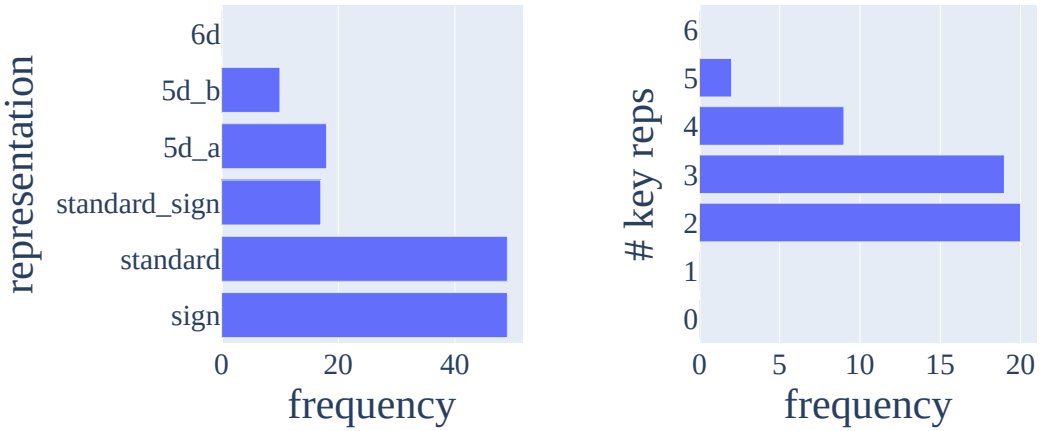

Figure 14: (Left) The number of times each representation is learned over 50 seeds, for S_5 trained on the MLP architecture. We see the 1d sign and 4d standard representations are most commonly learnt, standard_sign (4d), 5d_a and 5d_b are learnt approximately equally and less often, and 6d is never learned. (Right) The number of key representations of these 50 runs. Most commonly we have two key representations (typically sign and standard), but sometimes we learn more.

We naively thought that the complexity of a general representation would correlate with it's dimension[5]. For S_5, since the 4 dimensional representations are the lowest faithful representations, we expected representations of at most this dimension to be learned, and the model to choose arbitrarily between learning any one of them, or both. Emperically, we found this claim to be false. In particular, networks commonly learned higher dimensional representations, as can be seen in Figure 14. We also see in Figures 14 and 15 that the network preferred the standard representation over the standard_sign representation, when in fact standard_sign offers *better* performance for fixed weight norm.

While not deterministic, Figure 14 shows at least a probabilistic trend between our naive feature complexity and learning frequency, suggesting meaningful measures of feature complexity may exist. One complication here is that, as discussed in Section A.6, models are trading off weight against performance. Representations with more degrees of freedom may also offer better performance for fixed total weight norm, so which the model may prefer, and thus which is least complex, is unclear.

Number of representations learned varies. Across seeds, in addition to different representations being learned, we too find different *numbers* of representations are learned, also shown in Figure 14. This is surprising to us. We additionally find that Transformers consistently learn fewer representations than MLPs, despite having *more* parameters. We view this as further evidence against the strongest forms of circuit and feature universality, and suggests there is a degree of randomness in what solutions models learn.

Lower dimensional representations are generally (but not always) learned first. Under any reasonable definition of complexity, the 1d sign representation is simpler than other S_5 representations. Figure 15 shows that the sign representation is consistently learned first. While it is very easily learned, it also generalizes poorly. In contrast, higher dimensional faithful features are harder to learn but generalize better. These correspond to type 1 and type 3 patterns according to the taxonomy presented in Davies et al. (2022). We however *do not find evidence that all representations are learned in strict order of dimension*, against our naive hypothesis's predictions, further evidence against strong universality.

C Related Work

Mechanistic Interpretability and Universality. Our style of post-hoc interpretability in Section 4 is heavily inspired by the Circuits approach of Cammarata et al. (2020), Elhage et al. (2021) and Olsson et al. (2022). Universality, or *convergent learning*, was first proposed by Li et al. (2016), who

[5]In particular, we thought a reasonable definition would be the number of linear degrees of freedom in the $n \times d^2$ tensor of flattened representation matrices – i.e. the rank of the representation subspace of $\mathbb{R}^n$ (from Section A.2).

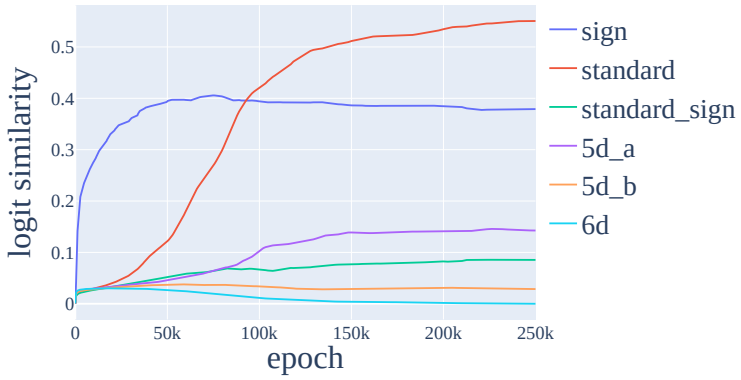

Figure 15: Mean evolution of logit similarity of each non trivial representation of S_5 over training averaged over 50 random seeds. We observe the sign representation is consistently learned early in training, and the standard representation is also often learned. Notably, the standard_sign representation is of comparable complexity to the standard representation, but learned to a lesser degree.

showed that many neurons are highly correlated with neurons in retrained versions of the same model. Recently, a number of papers have shown that networks tend to develop similar features (Raghu et al., 2017; Kornblith et al., 2019). Olah et al. (2020) demonstrate universality of curve detector and high-low frequency detector features in early layers of CNN vision models, while also showing the circuits implementing them are analogous. Bansal et al. (2021) show, through model stitching, features natural to one network may also be natural to others.

Group Theory. Group theoretic tasks have in the past been used to probe the capability of neural networks to perform symbolic and algorithmic reasoning. (Zhang et al., 2022) evaluate and fine tune real language models to implement group actions in context. Liu et al. (2022a) study how Transformers learn group theoretic automata.

Phase Changes and Emergence. Recent work has observed emergent behavior in neural networks: models often quickly develop qualitatively different behavior as they are scaled up (Ganguli et al., 2022; Wei et al., 2022). Brown et al. (2020) find that, while total loss scales predictably with model size, models' ability to perform specific tasks can change abruptly with scale. McGrath et al. (2022) find that AlphaZero quickly learns many human chess concepts between 10k and 30k training steps and reinvents human opening theory between 25k and 60k training steps.

Grokking. Grokking is a form of emergence, first reported by Power et al. (2022), who trained small networks on algorithmic tasks, finding that test accuracy often increased sharply, long after maximizing train accuracy. Liu et al. (2022b) construct further small examples of grokking, which they use to compute phase diagrams with four separate 'phases' of learning. Davies et al. (2022) unify the phenomena of grokking and double descent as instances of phenomena dependent on 'pattern learning speeds'. Our findings agree with Liu et al. (2022c) in that grokking seems intrinsically linked to the relationship between performance and weight norms; and with Barak et al. (2023) and Nanda et al. (2023) in showing that the networks make continuous progress toward a generalizing algorithm, which may be tracked over training by *progress measures*.

D Future Work

Below, we outline some areas for future work.

Further group theoretic tasks. In this work we focus on the task of group composition. Power et al. (2022) find that several other binary operations on pairs of input elements also grok, some of which are valid on any group. A trivial extension would be to the task $(a, b) \to ab^{-1}$, which may be solved simply by learning a permutation of the right embedding. A non trivial extension would be to conjugacy $(a, b) \to aba^{-1}$, which is of mathematical significance. Or to $(a, b, c) \to abc$. Each of these may be solved via similar representation theoretic algorithms, though we hypothesize would require two ReLU layers to implement two matrix multiplies. Another class of group theoretic tasks include those of group actions (a superset of group composition type tasks), where we expect representation theoretic algorithms to apply too. Extending to semigroups (arbitrary associative

multiplication tables) expands the set of tasks one could model, though there is no equivalent of representation theory for semigroups.

Further investigation of universality in algorithmic tasks. We raise many questions in Section 5 regarding which representations networks learn. Better understanding the learning rates and generalization properties of features offers a promising direction for future work in understanding network universality. Further understanding the probabilistic nature of which features are learned and at what time may too have future relevance. In particular, lottery tickets (Frankle & Carbin, 2019) may be present in initialized weights that could allow the learned features of a trained network to be anticipated *before training*.

More realistic tasks and models. In this work, we studied the behaviour of small models on group composition tasks. However, we did not explore whether our results apply to larger models that perform practical tasks. Future work could, for example, study universality in language models in the style of Olsson et al. (2022).

Understanding inductive biases of neural networks. A key question in the science of deep learning is understanding which classes of algorithms are natural for neural networks to express. Our work suggests that the GCR algorithm is in some sense a 'natural' way for networks to perform group composition (Appendix E). A more comprehensive understanding of the building blocks of neural networks could speed up interpretability work while helping us better understand larger models.

E Further Discussion on Inductive Biases

A key question in ML is of understanding the inductive biases of a network: what are the class of algorithms natural for a network to express? In addition to being useful across the board, results in this area could help guide hypothesis formation in mechanistic interpretability. Examples of preliminary work on understanding Transformer inductive biases is presented in Weiss et al. (2021) and Lindner et al. (2023).

Our work is useful in demonstrating the importance of *linearity* in networks. At a first glance, our algorithm seems an overly complex solution to the problem to us, requiring some advanced mathematics to understand. Yet, networks are extremely good at multiplying vectors of activations by matrices of parameters. Our algorithm consists mostly of these operations, with a single step of activation-activation multiplication in the middle to implement the matrix multiply $\rho(a), \rho(b) \to \rho(a)\rho(b)$. We discuss how networks may implement this operation in Appendix A.8. Note that the distinction between parameter-activation multiplication and activation-activation is important, with the former substantially easier for networks to implement. We also use a factored architecture (Appendix G), which results in a low-rank implicit bias, which may encourage a sparse number of representations to be learned. More subjectively, we found the process of reasoning through the algorithm and its implementation in the model to be insightful for better understanding networks ourselves.

We view this as evidence that the class of functions natural to humans and natural to networks are fundamentally different. Gaining examples like these is a step forward, but much more future work remains to be done in gaining a better understanding on these topics. Findings like these have in the past been beneficial in understanding real behaviour in networks, for instance the induction heads found by Olsson et al. (2022) are an important part of the circuit for indirect object identification found by Wang et al. (2022).

F Similarities and differences with prior work on reverse engineering modular addition

Here, we summarize the prior work of Nanda et al. (2023) that we build on, and detail where our experimental approaches differ. We note our contributions differ in that we use our mechanistic understanding to study universality. The authors train a one-layer Transformer model, of same type as we use (Section G) on modular addition. They find strong evidence it performs a completely understandable algorithm involving discrete Fourier transforms of the two inputs at various frequencies, and then makes use of various trigonometric identities to combine these. The key result which we generalize is that given inputs a and b the network computes $\cos(\omega(a+b-c))$ for each possible output z over some fixed set of frequencies ω. Taking the argmax of this expression over c gives the

correct answer. One can track the progress of this computation faithfully through the Transformers activations and weights.

Using this mechanistic understanding, the authors define the concept of a 'progress measure' that underlies the emergent behavior of grokking, a qualitative and discontinuous change in model behavior. They find that the training history of the model can be separated into three stages. First, the model memorizes the training data. Then, the circuit components for the general algorithm form smoothly. Finally, the memorized algorithm is cleaned up and removed as it is more complex and not favored by weight decay. Grokking occurs during cleanup, at the critical point after which the learned general algorithm is competitive with the 'memorized' algorithm – performance of the general algorithm is heavily hampered by 'noise' from the memorized algorithm. Crucially, the progress measures show that the components responsible for grokking arise before the sharp discontinuity in test loss.

We follow this approach closely. Our techniques in Section 4 are heavily inspired by Nanda et al.'s approach. Our precise analysis though differs substantially. Fourier transforms are elegant, but specific to the modular addition task. We instead work with representation matrices, and subspaces.

On modular addition of 113 elements, i.e. group composition on C_{113}, we are able to replicate their results in our framing. As discussed in Section 3, their algorithm maps precisely onto our GCR algorithm, and both approaches may be used to understand the cyclic group task. The mapping of their findings onto ours is fairly clear for embeddings, unembeddings and logits. For MLP neurons, they found that most neurons were well explained by a quadratic form of sinusoidal functions of the 9 terms within a single frequency. This quadratic form shared coefficients in such a way such that this had 2 redundant degrees of freedom, giving 7 terms. In our case, MLP neurons contain information pertaining to $\rho(a)$, $\rho(b)$ and $\rho(ab)$. In the special case of cyclic representations (see Appendix H.2.1), each of these terms has 2 degrees of freedom by antisymmetry. Adding a constant gives precisely the same seven terms.

G Architecture Details

Our mainline model is trained on 40% of all n^2 entries in the multiplication table of the group. We use full batch gradient descent. We use weight decay with $\lambda = 1$, and the AdamW optimizer, with learning rate $\gamma = 0.001, \beta_1 = 0.9$ and $\beta_2 = 0.98$. We perform $250,000$ epochs of training. As there are only n^2 possible input pairs, we evaluate test loss and accuracy on all pairs of inputs not used for training.

G.1 MLP

Our MLP architecture is summarized in Figure 1. Inputs a and b are encoded as n dimensional one-hot vectors. Each one-hot vector is embedded with $d = 256$. These are concatenated to form a 512 dimensional vector, which is fed into a $h = 128$ linear layer, with no bias term. [6] The output is mapped via an unembedding linear map, W_U, to n logits, corresponding to each of the n group elements. We did not tie the left embedding, right embedding or unembedding matrices. This is a simplified version of the Transformer architecture used by Nanda et al. (2023) (described below) which removes attention. Attention is both empirically irrelevant in this prior work, and not predicted to be necessary by our algorithm. The form of logits is therefore

```
Logits = W_U @ ReLU( W_MLP @ [W_left @ a, W_right @ b])
```

Note that the embedding matrices and linear layer have no non-linearity between them. When interpreting model calculations we will tie these matrices, and think of the a and b embeddings as the result of passing inputs through both layers. This methodology is inspired by (Elhage et al., 2021), [7]. The remainder of the operation of the linear layer is then to add these two 'total' embeddings and pass them through a ReLU. That is,

```
Logits = W_U @ ReLU( W_a @ a + W_b @ b)
```

where

[6] Emperically, we found adding a bias made little difference to our results, though we hypothesize that training models with a bias may improve the model's ability to perform matrix multiplication of activations, and hence interpretability.

[7] Here, the authors tie Transformer's Q and K matrices, and O and V matrices for the same reason.

```
W_a = W_MLP[:d, :] @ W_left        W_b = W_MLP[d:, :] @ W_right
```

G.2 TRANSFORMER

Our Transformer architecture for other runs is a decoder only architecture is based on Vaswani et al. (2017). It is identical to the set up for mainline experiments in Nanda et al. (2023). The input to the model is of the form "a b =", where a and b are encoded as n-dimensional one-hot vectors, and '=' is a special token above which we read the output c. We use a one-layer ReLU Transformer, token embeddings with d = 128, learned positional embeddings, 4 attention heads of dimension $d/4 = 32$, and $n = 512$ hidden units in the MLP. At points we analyze it's embedding W_E, MLP layer, and map to logits $W_L = W_U W_{out}$, ignoring the residual skip connection, which we find empirically is not utilized significantly for our tasks.

H MATHEMATICAL REPRESENTATION THEORY

In this section we present the results from group, representation, and character theory we make use of. Further details may be found in Alperin & Bell (1995). We begin by motivating our use of representation theory in this context. Groups are an abstraction of the idea of symmetry. In practice though, groups are not purely abstract objects, and tend to arise due to their action on other things. Often, these things are naturally attached to some vector space V, such that G gives rise to a linear action ρ on V, which we call a representation.

Theorem H.1. ***(Cayley)*** *Every group is isomorphic to a subgroup of a symmetric group.*

Definition H.2. A linear representation ρ is a group homomorphism $\rho : G \to GL(V)$ where $GL(V)$ denotes the general linear group of some vector space V over a field $\mathbb{F}$, the set of linear maps on V.

We focus on real representations, i.e. group homomorphisms $\rho : G \to GL(\mathbb{R}^d)$, the set of real invertible $d \times d$ matrices. We give some concrete examples of such representations of particular groups in Section H.2. We hypothesize representations are a natural way for a neural networks to implement operations on group elements. Representing group elements in a linear algebra theoretic manner seems like it would be advantageous to a networks natural operations of matrix multiplication and addition. We discuss this observation further in Appendix E.

Definition H.3. Let $\rho : G \to GL(V)$ be a linear representation. ρ is said to be **irreducible** if ρ has no G-stable subspace. That is, there is no subspace of V on which ρ defines a sub-representation of G.

From now on, we will use the term *irrep* to refer to irreducible representations. Irreps are the key object of interest. This is due to Maschke's Theorem.

Theorem H.4. ***(Maschke)*** *Every representation of a finite group G is a direct sum of irreducible representations. That is, there exists some basis in which all representation matrices are block diagonal, where the block sizes $d_1, \ldots, d_k$ are the same for all $\rho(g)$ with $g \in G$.*

Example. Every group G has a representation for any dimension d mapping each group element to identity matrix I_d. This is the direct sum of d one-dimensional irreducible representations named the 'trivial' representation, given by $\rho(g) = 1$ for all $g \in G$.

This representation isn't practically useful, as the network can not use these representations to perform calculations on group elements. We will often exclude the trivial representation and refer to *non-trivial irreps*. There are a finite number of these due to

Theorem H.5. *Let G be a group of order n and let ρ_i be distinct (up to isomorphism) irreducible representations of G over some splitting field $\mathbb{F}$. Let d_i be the dimension of ρ_i, and r be the number of irreducible representations. Then $n = d_1^2 + \cdots + d_r^2$.*

Some representations are more useful to the network than others.

Definition H.6. A representation ρ is said to be **faithful** if different elements g of G are represented by distinct linear maps $\rho(g)$. In other words, the group homomorphism $\rho : G \to GL(V)$ is injective.

Faithful representations are the most useful to the network, though we will often see networks also make use of lower degree non-faithful representations too.

Character theory forms an important part of representation theory, and will be important to our use case.

Definition H.7. Let V be a finite-dimensional vector space over a field $\mathbb{F}$ and let $\rho : G \to GL(V)$ be a representation of a group G on V. The **character** of ρ is the function $\chi_\rho : G \to \mathbb{F}$ given by $\chi_\rho(g) = \operatorname{tr} \rho(g)$, the trace of the representation matrix.

Character's are *class functions* – that is, they take a constant value on each conjugacy class of the group.

Theorem H.8. *Let G be a group, and $\rho : G \to GL(\mathbb{R}^d)$ a real representation of it of dimension d. For $g \in G$, $\chi_\rho(g) = \operatorname{tr} \rho(g) \leq d$ with equality iff $\rho(g) = I$.*

Proof. Let $|G| = n$. Since ρ is a group representation, and the order of elements in a group divide n, $\rho(g)^n = I$ for all g. The eigenvalues of $\rho(g)$ are therefore n'th roots of unity, so each character is a sum of roots of unity. By the triangle inequality, the claim holds. □

We now present some useful facts about characters.

Theorem H.9. ***(Schur's Orthogonality Relation of Characters)*** *The space of complex-valued class functions of a finite group G is endowed with a natural inner product, given by*

$$\langle \alpha, \beta \rangle = \frac{1}{|G|} \sum_{g \in G} \alpha(g) \overline{\beta(g)}$$

where $\overline{\beta(g)}$ denotes the complex conjugate. With respect to this inner product, the irreducible characters form an orthonormal basis for the space of class functions, yielding the orthogonality relation

$$\langle \chi_i, \chi_j \rangle = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}$$

Theorem H.10. ***(Schur's Orthogonality Relation of Matrix Elements)*** *Let ρ_λ be irreducible representations of a finite group G of dimension d_λ with matrix presentations Γ^λ_{ij}. Without loss of generality, we may assume Γ^λ is unitary, as any matrix representation is equivalent to a unitary representation.*

Then

$$\sum_{g \in G} \overline{\Gamma^\lambda(g)}_{ij} \Gamma^\mu_{i'j'} = \delta^{\lambda_\mu} \delta_{ii'} \delta_{jj'} \frac{|G|}{d_\lambda}$$

Note that the overbar denotes a complex conjugate, and the unitarity assumption only affects the constant, not the orthogonality.

H.1 Example

Example. The cyclic group C_n is generated by a single element r and naturally represents the set of rotational symmetries of an n-gon, where r corresponds to rotation by $2\pi/n$. This motivates a 2 dimensional representation – a set of n 2×2 matrices, one for each group element:

$$\rho(r^k) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

for element r^k corresponding to rotation by $\theta = 2\pi k/n$. This representation is irreducible, since there is no subspace of $\mathbb{R}^2$ on which the set of rotation matrices restricts – they each rotate the whole space. The character of each representation element is the trace $\chi_\rho(r^k) = 2\cos\theta$.

H.2 EXPLICIT GROUPS AND REPRESENTATIONS

Here, we describe the irreducible representation matrices for particular groups. Our methods for reverse engineering networks require mechanistic understanding of the precise form of representations. The classification of irreducible representations for any given group requires some machinery not presented here, and which we don't require for the purposes of our work. We just state the key results.

H.2.1 IRREDUCIBLE REPRESENTATIONS OF THE CYCLIC GROUP

The cyclic group C_n encodes rotational symettries of an n-gon. Over the reals, the irreducible representations of C_n fall into three classes. Note that Theorem H.5 does not apply here as $\mathbb{C}$ is a splitting field for C_n, but $\mathbb{R}$ is not.

1. the 1-dimensional trivial representation $\mathbf{1}$
2. the 1-dimensional sign representation $\mathbf{1_{sgn}}$, which only appears if the group order is even.
3. the 2-dimensional standard representations $\mathbf{2_k}$ of rotations in the Euclidean plane by angles that are integer multiples of $\frac{2\pi k}{n}$ for $k \in \mathbb{N}$, $0 < k < n/2$. The representation matrices may be written explicitly as

$$\rho_k(x) = \begin{pmatrix} \cos\left(\frac{2\pi k}{n}x\right) & -\sin\left(\frac{2\pi k}{n}x\right) \\ \sin\left(\frac{2\pi k}{n}x\right) & \cos\left(\frac{2\pi k}{n}x\right) \end{pmatrix}$$

Note the complex representations are much simpler, consisting of the n'th roots of unity. The sign representation appears then due to -1 being a root of unity iff n even. For $k = n/2$, the 2d representation is the direct sum of two copies of the sign representation, so is not irreducible, and for $k > n/2$ we have the isomorphism $\mathbf{2_{n-k}} \backsimeq \mathbf{2_k}$.

H.2.2 IRREDUCIBLE REPRESENTATIONS OF THE DIHEDRAL GROUP

We focus on dihedral groups $D_n = \langle r, s | r^n = s^2 = e, srs = r^{-1} \rangle$, with n odd. These encode all symmetries of an n-gon, rotational and reflectional. The representations of these groups are much the same as those of cyclic groups, and fall into three categories.

1. the 1-dimensional trivial representation $\mathbf{1}$
2. the 1-dimensional sign representation $\mathbf{1_{sgn}}$, mapping $\langle r \rangle$, i.e. rotations, to 1, and the coset, i.e. reflections to -1.
3. the 2-dimensional standard representations $\mathbf{2_k}$, corresponding to rotations and reflections in the Euclidean plane.

$$\rho_k(r^l) = \begin{pmatrix} \cos\left(\frac{2\pi k}{n}l\right) & -\sin\left(\frac{2\pi k}{n}l\right) \\ \sin\left(\frac{2\pi k}{n}l\right) & \cos\left(\frac{2\pi k}{n}l\right) \end{pmatrix}$$

$$\rho_k(r^l s) = \begin{pmatrix} \cos\left(\frac{2\pi k}{n}l\right) & \sin\left(\frac{2\pi k}{n}l\right) \\ \sin\left(\frac{2\pi k}{n}l\right) & -\cos\left(\frac{2\pi k}{n}l\right) \end{pmatrix}$$

H.2.3 IRREDUCIBLE REPRESENTATIONS OF THE SYMMETRIC GROUP

Our mainline experiments involve the permutation, or symmetric, group of 5 elements, denoted S_5. We denote general permutation groups of n elements S_n.

Table 5: The lowest degree irreps for S_n for $n \geq 7$, and their dimension. For $n \leq 7$, additional symmetries give rise to other low dimensional irreps on top of these.

S_n IRREP	DIMENSION
TRIVIAL	1
SIGN	1
STANDARD	$n-1$
STANDARD $\otimes$ SIGN	$n-1$

The irreps in Table 5 may be fairly easily constructed. We constructed trivial irreps in Appendix H.

The sign representation are a set of 1×1 matrices representing a kind of parity. Permutations may be decomposed as a (non unique) sequence of swaps. The parity of this number of swaps is in fact well defined, and defines a subgroup of the symmetric group named the alternating group. Mapping this alternating group to $+1$, and the coset to -1 gives the sign representation. In general, any group containing a subgroup of index 2 is naturally endowed with a sign representation in a similar manner.

Next is the standard representation. This is essentially the set of permutation matrices – $n \times n$ square binary matrices, with only one 1 in each row and column, and 0s elsewhere. This representation has dimension n, though, not $n - 1$. This is because it turns out to be reducible. Recalling Definition H.3, this has an invariant subspace under the action of G, spanned by the vector sum of all basis elements. The irreducible representations recovered are the standard and trivial representations.

Standard $\otimes$ Sign denotes the tensor product of the standard and sign representations, which is just their matrix product as the sign representation is 1 dimensional.

S_5 has three higher degree representations, which we denote 5d_a, 5d_b, 6d. We omit their construction here.

H.2.4 Irreducible Representations of A_5

As a subgroup of S_5, A_5 inherits representations from S_5. However, the six dimensional representation of S_5 becomes reducible, splitting into two three dimensional irreps of A_5. We omit details here.

I Universality Data

Here, we give full, unaveraged summary statistics of our runs on 4 seeds discussed in Section 5. We omit the 50 MLP S_5 runs.

Why are logit FVE scores relatively low? We note that our algorithm's prediction does not explain all of the logits, as can be seen from FVE being less than 100%. Here, we provide some discussion on this. We do not believe this is evidence against our claim that we completely understand the important algorithms our network is implementing. Rather, we believe this is a side product of limitations of our architecture. In particular, as we discuss in Section A.3 and Appendix A.8, the model uses the ReLU activations to implement matrix multiplication of $\rho(a)$ and $\rho(b)$ to $\rho(ab)$. This can only be approximated by ReLUs, and produces other terms, notably significant components of $\rho(a)$ and $\rho(b)$. This means that the network cannot perfectly extract the components of $\rho(ab)$ because they do not correspond to directions orthogonal to all other terms, resulting in the map to logits W_U extracting other terms and non-character terms being present in the logits.

Table 6: Results from MLP runs on various groups and seeds. Our algorithm is universally learned. Key representations are listed in order learned.

			Loss			FVE					
Group	Seed	Key Representations	Test	Exc.	Res.	Logit	W_a	W_b	W_U	MLP	$\rho(ab)$
C_{113}	1	32, 50, 44, 16, 17, 34, 4, 11, 22, 1, 8, 13, 25	2.86e-05	6.22	9.84e-03	93.68%	99.23%	99.07%	97.95%	91.08%	11.80%
C_{113}	2	7, 22, 25, 24, 36, 30, 14, 41, 44, 48, 50	5.89e-06	5.15	1.96e-06	98.08%	99.65%	99.81%	98.35%	89.95%	11.71%
C_{113}	3	37, 14, 55, 47, 52, 34, 9, 5, 54, 45, 2, 3, 18, 28, 39	2.46e-05	5.69	8.29e-03	92.69%	99.57%	98.96%	97.85%	90.01%	12.14%
C_{113}	4	55, 11, 30, 27, 43, 34, 29, 22, 3, 53	6.15e-06	6.74	9.38e-03	97.89%	99.65%	99.72%	98.06%	89.98%	12.46%
C_{118}	1	31, 49, 1, 12, 22, 45, 2, 20, 24, 28, 44, 56	5.75e-06	8.53	3.90e-06	97.22%	99.55%	99.55%	98.46%	94.61%	12.66%
C_{118}	2	47, 23, 19, 29, 16, 44, sign, 30, 32, 38, 46, 58	5.19e-06	7.52	2.90e-03	98.25%	99.88%	99.81%	98.36%	94.67%	13.59%
C_{118}	3	16, 30, 39, 8, 43, 48, sign, 11, 32, 40, 58	5.51e-06	10.92	7.93e-03	96.84%	99.84%	99.77%	98.40%	99.00%	13.63%
C_{118}	4	sign, 14, 5, 25, 57, 1, 22, 2, 4, 10, 28, 50	5.11e-06	7.90	3.56e-03	98.19%	99.73%	99.82%	98.49%	95.09%	13.17%
D_{59}	1	19, 20, 15, 6, 14, 8, 3, 7, 12, 16, 21, 29	9.50e-06	9.17	7.41e-07	48.65%	99.46%	99.40%	98.58%	86.81%	11.65%
D_{59}	2	18, 10, sign, 19, 6, 26, 7, 20, 21, 23	4.30e-06	12.74	1.77e-06	54.92%	99.90%	99.93%	98.57%	88.05%	12.66%
D_{59}	3	sign, 20, 22, 16, 9, 12, 11, 15, 18, 19, 24	6.88e-06	11.06	1.91e-06	56.79%	99.57%	99.71%	98.50%	87.82%	13.05%
D_{59}	4	sign, 7, 10, 15, 21, 17, 19, 20, 29	4.68e-06	16.52	1.98e-06	53.05%	99.90%	99.89%	98.42%	88.03%	12.41%
D_{61}	1	sign, 19, 23, 6, 7, 3, 24, 5, 12, 13, 14, 15	2.33e-05	11.21	1.76e-06	51.46%	99.17%	99.63%	98.05%	87.71%	12.05%
D_{61}	2	sign, 4, 29, 8, 27, 26, 19, 28, 14, 9, 2, 7, 3, 16, 18	2.87e-05	10.20	1.48e-06	53.04%	99.30%	99.05%	98.44%	87.15%	13.00%
D_{61}	3	15, 14, 9, 26, 2, 25, sign, 28, 4, 18, 30	5.58e-06	14.86	1.74e-06	54.99%	99.68%	99.89%	98.28%	88.44%	12.60%
D_{61}	4	20, 21, 19, 7, 17, 15, 23, sign, 14, 27, 30	1.39e-05	11.75	1.77e-06	50.33%	98.90%	99.24%	98.28%	87.15%	12.26%
S_5	1	sign, standard-sign, standard, 5d-a	3.14e-05	10.09	1.52e-07	39.05%	100.00%	99.96%	94.38%	87.95%	10.53%
S_5	2	sign, standard	2.94e-06	7.59	7.08e-07	84.81%	100.00%	100.00%	94.05%	88.88%	12.97%
S_5	3	sign, standard, 5d-b	4.32e-06	11.97	2.17e-08	59.89%	100.00%	99.99%	94.97%	88.85%	12.38%
S_5	4	sign, standard	2.25e-06	17.21	1.96e-09	59.25%	100.00%	100.00%	93.18%	89.95%	12.66%
S_6	1	5d-b, standard-sign, 5d-a, standard	5.12e-05	12.97	1.98e-06	34.50%	99.77%	99.87%	93.25%	86.69%	8.38%
S_6	2	sign, standard, 5d-b	1.36e-05	13.42	2.52e-07	64.15%	100.00%	100.00%	93.42%	87.05%	10.27%
S_6	3	sign, standard-sign, 5d-a, 5d-b, standard	9.09e-05	10.86	6.87e-06	40.97%	98.96%	99.42%	94.42%	84.15%	7.52%
S_6	4	sign, 5d-b, standard-sign, standard	4.21e-05	11.41	1.54e-06	56.96%	99.86%	99.83%	93.60%	87.64%	9.75%
A_5	1	standard, 3d-a, 3d-b	6.27e-05	7.23	1.56e-06	51.38%	98.52%	98.69%	93.08%	84.13%	9.46%
A_5	2	standard, 3d-a, 3d-b	5.09e-06	9.45	3.96e-07	43.86%	98.99%	99.08%	92.94%	85.11%	10.62%
A_5	3	3d-a, 5d-a, standard	3.73e-06	11.55	5.70e-08	49.96%	99.53%	99.74%	92.73%	89.12%	10.81%
A_5	4	5d-a, 3d-a, standard, 3d-b	5.93e-06	11.03	9.81e-08	45.57%	99.14%	99.73%	94.32%	88.39%	10.14%

Table 7: Results from Transformer runs on various groups and seeds. Our algorithm is universally learned. Key representations are listed in order learned.

			Loss			FVE				
Group	Seed	Key Representations	Test	Exc.	Res.	Logit	W_E	W_U	MLP	$\rho(ab)$
C_{113}	1	16, 30, 56	1.88e-07	9.77	2.26e-02	96.85%	90.08%	99.49%	92.67%	16.05%
C_{113}	2	43, 53, 52, 49	3.89e-07	8.45	1.45e-02	96.70%	96.91%	99.71%	89.72%	17.17%
C_{113}	3	25, 56, 33, 19	3.39e-07	8.76	8.19e-03	95.21%	95.70%	99.23%	93.32%	16.23%
C_{113}	4	11, 12, 18	1.53e-07	10.69	3.96e-02	97.93%	98.05%	99.64%	92.77%	17.62%
C_{118}	1	37, 10, 16, sign, 19	1.67e-07	9.54	1.63e-03	98.55%	93.88%	99.82%	94.81%	17.54%
C_{118}	2	8, 12, 27, 57	1.94e-07	12.67	2.42e-03	98.35%	98.12%	99.49%	92.76%	16.36%
C_{118}	3	53, 51, 4, 46	1.74e-07	6.25	3.16e-03	98.59%	92.74%	99.84%	93.49%	14.48%
C_{118}	4	17, sign, 29	1.59e-07	35.28	1.01e+00	97.29%	91.46%	99.42%	97.46%	20.05%
D_{59}	1	sign, 21, 5, 2	7.83e-06	54.66	1.06e-04	46.36%	98.36%	95.46%	85.38%	11.30%
D_{59}	2	1, 15, 23	3.76e-06	69.84	6.62e-08	51.28%	98.10%	99.80%	84.26%	10.00%
D_{59}	3	22, 20, 26	4.21e-07	31.95	6.76e-06	67.51%	99.12%	99.36%	85.09%	10.57%
D_{59}	4	1, 16, sign, 24, 4	8.07e-07	29.24	1.23e-07	51.58%	98.76%	99.49%	85.31%	11.53%
D_{61}	1	13, 26, 6, 16, 4, 1, 14, 12, 18	6.50e-02	27.57	3.41e-03	59.47%	95.48%	95.39%	86.08%	10.62%
D_{61}	2	sign, 24, 4, 18	9.56e-06	51.42	3.80e-01	40.88%	98.91%	94.59%	85.70%	11.50%
D_{61}	3	8, sign, 23, 28	4.23e-07	53.44	6.87e-08	51.71%	99.06%	99.82%	85.38%	11.30%
D_{61}	4	2, 6, 13	1.89e-07	34.13	7.29e-08	56.37%	99.88%	99.79%	85.20%	11.04%
S_5	1	sign, standard-sign	1.42e-07	16.63	1.78e-09	62.85%	99.86%	99.98%	80.51%	8.44%
S_5	2	sign, standard-sign	2.41e-07	12.66	1.60e-08	73.46%	99.69%	99.92%	80.85%	7.58%
S_5	3	sign, standard	9.39e-08	20.86	1.73e-11	59.06%	99.91%	99.99%	89.87%	12.44%
S_5	4	sign, standard	9.51e-08	20.93	1.77e-11	59.07%	99.90%	99.99%	89.88%	12.46%
S_6	1	5d-b	2.26e-06	531.31	4.93e-16	44.50%	99.97%	100.00%	88.06%	9.57%
S_6	2	sign, 5d-b	2.91e-06	62.67	4.19e-06	65.69%	99.86%	99.74%	80.58%	7.90%
S_6	3	sign, 5d-b	1.82e-06	286.64	9.78e-12	49.53%	99.96%	100.00%	88.32%	9.96%
S_6	4	sign, 5d-b	1.87e-06	286.08	1.02e-11	49.58%	99.96%	100.00%	88.32%	9.96%
A_5	1	3d-a, 3d-b	1.35e-07	15.65	1.40e-03	63.19%	94.62%	95.00%	77.00%	6.30%
A_5	2	3d-a, 3d-b	1.30e-07	15.52	1.68e-03	63.29%	94.55%	94.95%	77.08%	6.33%
A_5	3	5d-a, 3d-b, 3d-a	1.95e-01	27.92	1.30e-10	25.23%	101.04%	99.68%	90.65%	8.52%
A_5	4	standard	8.06e-08	19.95	1.21e-11	52.84%	99.92%	99.98%	89.52%	11.75%