Probing the effect of data representation on transformer learning

Sarah Scullen SARAH.SCULLEN@PNNL.GOV
Pacific Northwest National Laboratory
Henry Kvinge HENRY.KVINGE@PNNL.GOV
Pacific Northwest National Laboratory
University of Washington
Helen Jenne HELEN.JENNE@PNNL.GOV
Pacific Northwest National Laboratory

Abstract

The choice of data representation can affect what a model learns, motivating the need for systematic methods to study this effect. Permutations and their statistics provide a controlled setting in which the underlying object is fixed and only its input representation varies. In this paper, we train small transformers on multiple representations of the same permutations and analyze them using linear probes, attention head ablations, and representational similarity measures. We find that convergence across input representations is task-dependent: models trained to compute the length of the longest increasing subsequence from different representations converge on similar internal computations, while models trained to compute other statistics show more varied behavior, including representation-specific algorithms. Beyond these specific findings, this work provides a controlled testbed for future interpretability research studying when models trained on different input representations learn the same mechanism.

Keywords: Permutations, Transformer, Mechanistic Interpretability, Representational Similarity

1. Introduction

It is well-known that the choice of data representation, how an input is encoded and passed to the model, can greatly influence learning outcomes. In natural language processing, the choice between word-, subword-, byte-, and character-level encodings can substantially affect downstream performance and robustness (2; 18; 25). The encoding scheme used to represent real numbers affects how transformers learn arithmetic tasks and matrix operations (4; 20). Other examples arise in molecular learning, where a molecule can be represented as a SMILES string or graph (27), and in audio processing, where a signal can be represented as a raw waveform or time-frequency features (23). Despite these observations, we lack controlled settings for studying how different encodings of the same underlying object affect task difficulty, model performance, and the algorithms that models discover.

In this paper, we use permutations and their statistics to study how data representation affects learning. Permutations admit many natural representations, each making different structural properties explicit. They also support numerous statistics, yielding a range of potential classification tasks. Studying pairwise combinations of these representations and tasks on the same mathematical objects gives a controlled way to examine how representational choices impact learning. In particular, we are interested in the following questions.

- How does the data representation affect task difficulty and the number of examples it takes for the model to learn?

- When models solve the same task from different representations, do they rely on similar attention-head circuits or develop similar internal representations?

- For tasks natural in one data representation but not another, do models given a less convenient representation reconstruct the task-natural representation internally, or instead learn task-specific summaries or abstract permutation features?

To study these questions, we consider four permutation statistics and three data representations, training a separate small transformer for each (representation, statistic) pair. With sufficient data, models reach very high accuracy on nearly all pairs, with data representation substantially affecting sample complexity. We therefore focus not on whether models learn, but on how data representation affects sample complexity and the learned computation.

The learned computations can be studied from several complementary perspectives. Representational similarity measures compare how information is geometrically organized by comparing the activations of intermediate layers (12). Linear probes test whether a specific property is linearly decodable from those activations (1). Interventional methods such as attention-head ablation test whether a model component contributes causally to a model behavior, whereas circuit discovery additionally seeks to explain how components interact to implement that behavior (8; 21). These approaches support different claims and each have limitations. For example, some representational similarity metrics are sensitive to outliers and model size (7; 9); high probe accuracy may reflect the expressive power of the probe; and mechanistic interpretations depend on intervention choices and can admit competing explanations (19; 35; 39). We therefore combine methods from each perspective to support our claims about what the models have learned.

The extent to which models trained on different input data representations converge depends on the task. We present examples in which models trained on different representations develop similar internal computations, along with cases that show more representation-specific or mixed behavior. Beyond discussing how small transformers compute specific permutation statistics, this work provides a controlled testbed for future mechanistic interpretability research on when models discover shared mechanisms, and when differences in data representation lead to different internal algorithms.

2. Background

We begin by introducing permutations and their different input notations (see Stanley (29) for a thorough introduction). For the remainder of this paper, we will use the term *input notation* (rather than data representation) for the external encoding of a permutation, reserving *representation* for a model's internal activations. We then introduce the tasks: the permutation statistics that transformers are trained to predict.

2.1. Permutations: notations

A *permutation* σ is a bijective function from the set $[n] := \{1, 2, \ldots, n\}$ to itself. The set of all permutations of $[n]$, denoted $\mathcal{S}_n$, forms the symmetric group, a central object in algebra, combinatorics, and representation theory. The *one-line notation* of a permutation is the vector $(\sigma(1), \sigma(2), \ldots, \sigma(n))$. For example, the permutation that maps $1 \mapsto 2$, $2 \mapsto 5$, $3 \mapsto 3$, $4 \mapsto 4$, and $5 \mapsto 1$ is represented in one-line notation by the sequence 25341.

While one-line notation is common, mathematicians also use several other notations that encode different structural aspects of permutations. Each permutation can be written as a product of disjoint *cycles*. In *cycle notation*, every integer in a cycle is mapped to the next integer in that cycle, with the last integer mapped back to the first. The permutation 25341 is written using its cycle notation as (125) or (125)(3)(4). We will use the former, omitting cycles of length one (called *fixed points*).[1] Cycle notation is not unique; in our datasets, each cycle is written starting with its smallest element and then cycles are ordered based on their smallest elements. Cycle notation makes some group-theoretic quantities (such as the order of the permutation as a group element) easy to compute.

An *inversion* of a permutation is a pair (i, j) where $i < j$ but $\sigma(i) > \sigma(j)$. In the permutation 25341, $(2, 3)$ is one of six inversions. The *(value-indexed) inversion vector* is the vector where $v_i = \#\{j > i : \sigma^{-1}(j) < \sigma^{-1}(i)\}$, so the ith entry counts how many values larger than i appear before it in one-line notation. For the permutation 25341, the inversion vector is $(4, 0, 1, 1, 0)$. Note that the one-line notation can be recovered from the inversion vector. The sum of the entries of the inversion vector is the *Coxeter length* of the permutation, important in the representation theory of the symmetric group.

2.2. Tasks

There are hundreds of permutation statistics that have been studied by mathematicians. Our choice of statistics was motivated by our goal to study whether models given inconvenient notations reconstruct the task-natural notation. Apart from one control task (parity) we chose statistics that are natural for at most one input notation, and therefore nontrivial to compute from at least two of the three notations.

Parity. The *parity* of a permutation is the parity of its Coxeter length. It can be computed from the inversion vector by summing its entries mod 2, from one-line notation by counting the number of inversions mod 2, and from cycle notation by counting the number of even-length cycles mod 2.

Why this task: This task serves as a control. Because parity has a straightforward computation in each notation, success on this task does not require models trained on different notations to learn a shared intermediate representation.

Length of the longest increasing subsequence (LIS). The length of the LIS of a permutation is the largest k for which there exist indices $i_1 < i_2 < \cdots < i_k$ with $\sigma(i_1) < \sigma(i_2) < \cdots < \sigma(i_k)$. For example, the length of the LIS of the permutation 25341 is 3. This task is most readily computed from one-line notation, but it is not trivial. It can be solved with dynamic programming or the *patience sorting algorithm* (Appendix B). From cycle notation or the inversion vector, a natural first step is to recover the one-line notation.

1. In our experiments, n is fixed, so omitting fixed points does not make this notation ambiguous.

Why this task: Computing the LIS is natural from one-line notation but not the other two notations. Do models trained on the inversion vector and cycle notation convert to one-line notation, or do they use a different strategy?

Number of cycles. We define the number of cycles as the count of cycles in the disjoint cycle decomposition of the permutation, excluding fixed points (24; 31). In our canonical cycle notation, this can be computed by simply counting the displayed cycles. This statistic is not immediately visible from one-line notation or the inversion vector.

Why this task: This task is trivial from cycle notation, but not from the other notations. Do the models trained on one-line notation and inversion vector build cycle-like features?

Number of topologically connected components of the chord diagram of a permutation. The *chord diagram* of a permutation is obtained by placing the labels $1, \ldots, n$ in order on a circle, and drawing a chord between i and $\sigma(i)$ for each label i. We count the connected components of the union of the chords and their endpoints, excluding the arcs of the boundary circle (3). Two chords that cross are in the same component. A fixed point $i = \sigma(i)$ contributes a single component. When there are no crossings in the chord diagram, the number of topologically connected components is equal to the number of cycles including fixed points.

Why this task: This task differs from the other three because it requires constructing a new intermediate object: the chord diagram. It is not immediately obvious which notation makes this statistic easiest to compute: the chords are easily read off from one-line notation, while cycle notation gives the connected components before crossings are taken into account.

3. Methods

3.1. Data and Model Architecture

We use all $8! = 40,320$ permutations in $\mathcal{S}_8$, with a random 70/15/15 train/validation/test split shared across all tasks and input notations. We train a decoder-only causal transformer, implemented using the `x-transformers` library (34), to predict the answer as the next token after an `[ANS]` delimiter. The loss is computed only on the answer token.

We use a symbolic tokenizer. Token IDs $0, \ldots, 4$ are reserved for the padding token, the `[ANS]` delimiter, and the punctuation symbols {`(`, `)`, `,`}; each integer k is encoded as $k + 5$. Inputs and answers share the same vocabulary. For all statistics considered here, each answer is a single token, and no answer value exceeds 8. We use comma tokens to separate integers in all input notations, while parentheses are used only in cycle notation. We chose hyperparameters to keep models small enough for interpretability analyses while still large enough to achieve high validation accuracy. This paper focuses on experiments with a 4-layer, 4-head transformer ($d_{\text{model}} = 128$). Appendix C contains further details on the model architecture and hyperparameter choices.

We considered several alternative design choices, including separate input and output vocabularies, classifier-head prediction instead of autoregressive next-token prediction, and encoder-only rather than decoder-only architectures. Ultimately, we opted for an architecture and training methodology analogous to supervised fine-tuning of language models. This choice reflects our goal of using a broadly applicable approach that can extend to other

studies of how input representation affects learning, rather than a specialized architecture tailored to permutation statistics.

3.2. Analysis

To understand what the trained models compute, we combine several analyses. Linear probes ask when in the forward pass a statistic becomes linearly decodable, whereas centered kernel alignment (CKA) and orthogonal Procrustes instead compare the geometry of the activation spaces. Attention-pattern summaries describe which token types are attended to at the answer-readout position. Ablations add a causal perspective, asking which attention heads are behaviorally important. Finally, for models trained on cycle notation, we test out-of-distribution generalization to alternative valid cycle-notation conventions.

Linear probing We train linear probes to test whether task statistics and auxiliary permutation-structure variables are linearly decodable from the residual stream. For each model, we save residual-stream activations at the answer-readout position across the held-out test set after the input embedding and after each transformer layer. We use an 80/20 stratified split of those activations to fit and evaluate an ℓ_2-regularized logistic regression classifier for each target. We average probe accuracies over 10 model seeds for each (notation, task) pair.

Linear probes for one-line notation: We fit eight independent 8-class probes per (model, layer), one for each $\sigma(i)$. Reported accuracy is the mean of the eight per-position accuracies.

Linear probes for characteristics of cycle notation: Training linear probes to predict the cycle notation string directly is complicated by its encoding: it contains parentheses, and the same permutation has multiple valid encodings. We instead probe two cycle-structure summaries: (i) the partition class (also known as the cycle type), which is the integer partition given by the cycle lengths, encoded as a single 22-class target; and (ii) the *per-integer cycle length*, the vector where entry i is the length of the cycle containing i.

Representation similarity For each task, we compare residual stream activations at the answer-readout position across models trained on different input notations using linear centered kernel alignment (CKA) (14) and orthogonal Procrustes (26). CKA compares centered linear-kernel similarity structure across examples, while Procrustes measures alignment after an optimal orthogonal transformation; precise definitions are given in Appendix C.

Attention Pattern Analysis Forward hooks on each attention module capture post-softmax attention weights over the held-out test set. We aggregate from the answer-readout query row, reporting two summaries per (layer, head): (i) attention mass directed to each input-token category (left/right parenthesis, comma, or integers) (ii) attention mass directed to each individual token id.

Attention head ablation We identify attention heads that are causally necessary for the learned computation by systematically zeroing out each head's contribution (8; 33). For each ablated model, we compute the *ablation effect*, the test-accuracy drop relative to the unablated baseline. Sorting heads within each layer by increasing ablation effect and averaging across seeds yields one importance vector per (notation, task) pair. We then compute the Spearman rank correlation (ρ) between importance vectors for each notation pair and

target statistic, where high similarity suggests similar layer-wise localization of attention-head importance. Full definitions and the seed-consistency analysis are in Appendix C.

Out-of-distribution generalization for cycle notation models As discussed in Section 2.1, there are multiple equivalent ways to write a permutation in cycle notation. To better understand how the models trained on the canonical cycle notation learn the tasks, we evaluate each cycle notation model on the same held-out permutations encoded using alternative valid conventions; details are given in Appendix C. This indicates whether model performance generalizes to these notation variants.

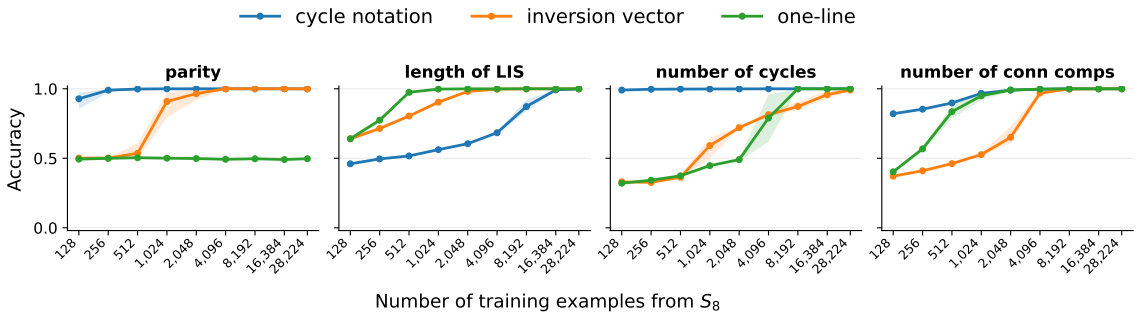

Figure 1: **Sample Complexity:** Curves show mean test accuracy over three random seeds. Shaded regions show the minimum-to-maximum range across seeds.

4. Results

When trained on all permutations in $\mathcal{S}_8$, small transformers achieved over 99% test accuracy on each task, regardless of input notation, except for learning parity from one-line notation. Some (notation, task) pairs required substantially less data than others to achieve this accuracy. Figure 1 shows test accuracy for each pair as training data increases on a logarithmic scale. These results align with our intuition about task difficulty: for parity, length of longest increasing subsequence, and number of cycles, models trained on one notation required less data than the others, likely because the task is trivial or more naturally computable from it. Computing the number of topologically connected components was the only task requiring the construction of a new intermediate object, and it was also the only task for which no notation required less data than the others.

We organize our analyses of the learned computations by task, recalling each task's motivating question before presenting results. Our methods were not equally informative. CKA and orthogonal Procrustes increase (resp. decrease) sharply once the answer becomes linearly decodable in both models, making high (resp. low) values difficult to attribute to similar intermediate structure. We therefore rely primarily on linear probes, attention-head ablation, and attention pattern analysis, leaving the representational similarity plots to the Appendix.

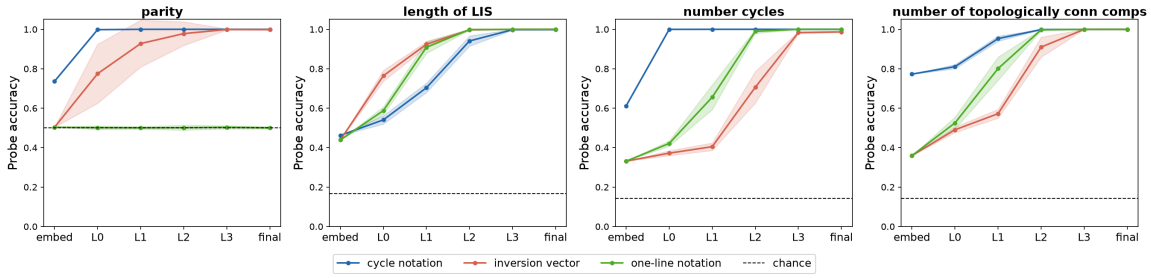

Figure 2: Linear probe accuracy by task (averages over 10 seeds).

4.1. Parity

The parity task serves as a control: a distinct algorithm is available from each notation, so we did not expect models trained on different notations to converge on similar internal representations or to develop features of the other notations. As noted above, models trained on one-line notation never exceeded chance accuracy on the parity task. Parity is known to be a difficult task for transformers (40), and we did not find different learning outcomes in hyperparameter sweeps or with more training data (all permutations from S_9).

For the two notations from which the model did learn the task, the cycle notation and inversion vector, our analyses indicate divergent, notation-specific computations. Attention maps for the cycle-notation models support a notation-specific algorithm: these models attend primarily to structural tokens. Specifically, a single head in layer 0 attends strongly to right parentheses; while heads in layers 2 and 3 attend to left parentheses, suggesting that an early head locates cycle ends, and later heads locate the corresponding cycle starts. (Matched start and end positions give cycle lengths, and the parity of the number of even-length cycles determines the parity of the permutation). The fact that accuracy was not affected by the cycle convention used to encode the input (Figure 4 in Appendix D) further supports reliance on structural tokens. Our methods do not reveal exactly what the inversion-vector models compute, but cycle features were not linearly decodable from their representations, suggesting they did not learn a cycle-based algorithm.

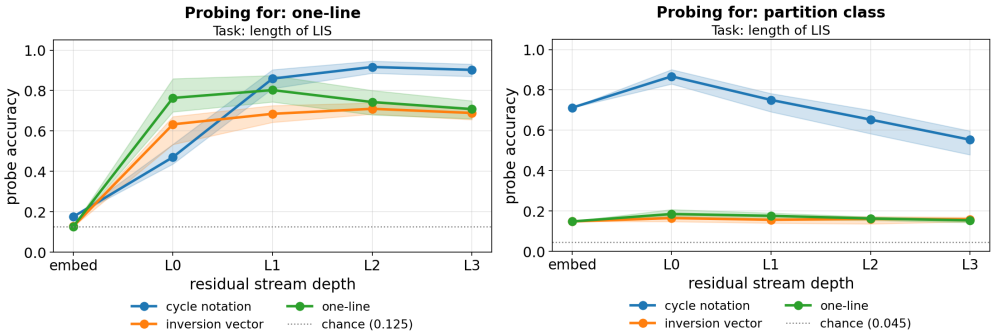

Figure 3: Linear probes for target one-line notation (left) and cycle type (right) for models trained to compute the length of the longest increasing subsequence.

4.2. Length of longest increasing subsequence

The length of the LIS task is most natural, though still nontrivial, to compute from one-line notation, so this task asked whether models trained on the inversion vector and cycle notation convert to one-line notation. *Our analyses suggest that LIS models converge toward similar internal computations across the three input notations, with cycle-notation models developing a one-line-like internal code.*

For all three notations, LIS probes are near chance at early layers and become accurate only later in the forward pass (Figure 2); so the answer is not available as an early-layer linear readout at the answer position. After the second layer, the LIS-probe trajectories for the one-line and inversion-vector models are nearly identical, as are their one-line probe accuracies, including when resolved by position rather than averaged (Figure 9 in Appendix D).

The strongest evidence that the cycle-notation models develop a one-line like code comes from linear probing: one-line notation probes decode the values $\sigma(i)$ with high accuracy at later layers (shown in Figure 3). In contrast, cycle-notation models trained on other tasks reach only 40% one-line probe accuracy (Figure 10 in Appendix D). Cycle-type probe accuracy also decreases after the first layer, in contrast to the other tasks, where it increases. The cycle-notation models thus appear to transform away from an explicit cycle-type representation. Sorted ablation profiles are likewise highly correlated across input notations (0.97 between cycle notation and one-line models; 0.93–0.94 for the other pairs).

Cycle-notation LIS models do not generalize to alternative valid cycle conventions. Our canonical form orders cycles and their elements by smallest element. A model can reconstruct σ while relying on that convention, so this failure is consistent with either a convention-dependent conversion to a one-line-like code or a convention-specific algorithm.

4.3. Number of cycles

The number of cycles is trivially readable from cycle notation, so this task tested whether models trained on one-line notation and inversion vectors develop cycle-like internal features. *Our analyses suggest that cycle-notation models solve this task directly by counting structural tokens, while one-line and inversion-vector models develop some cycle-relevant features but likely do not fully convert to a cycle-notation-like representation.*

From cycle notation, the number of cycles can be obtained by counting left (or right) parentheses. Consistent with this strategy, in cycle-notation models the number of cycles becomes linearly decodable after one layer (Figure 2), attention concentrates on structural tokens, and accuracy is unaffected by the cycle convention used to encode the input.

For the models trained on the other two notations, the number of cycles becomes linearly decodable after three layers (one-line) and four layers (inversion vector). Partition class (cycle-type) probes reach ≈74% accuracy for one-line models, and ≈65% for inversion-vector models, well above the 14.3% majority-class baseline, consistent with the development of cycle-structure-like features. However, this does not establish that either model literally converts to cycle notation. One-line information remains linearly decodable in the one-line model, and one-line probe trajectories are similar for one-line and inversion-vector models (Figure 10), suggesting that position-value information is retained or reconstructed while cycle-relevant summaries are added. This task therefore provides a contrast with the

LIS task, where cycle-notation models converged toward a one-line-like code; here neither notation converges toward a cycle-like one.

4.4. Number of topologically connected components

This task requires constructing a new intermediate object, the chord diagram, and it is not obvious a priori which input notation makes this construction easiest. *We found little evidence that models trained on all three notations converge to a common computation.*

Partition class (cycle type) probes are highly accurate in cycle-notation models, but much less so in the one-line and inversion-vector models, though still above the majority baseline (Figure 12). This suggests that the inversion-vector and one-line models learn some cycle-relevant information without making the full cycle structure linearly available. This is unsurprising since cycle structure is relevant to the task (when the chord diagram has no crossings, the number of components equals the number of cycles including fixed points). The ablation profiles show that the inversion-vector model consistently relies on a high-impact late-layer head, distinguishing it from the other two models. The cycle-notation and one-line models have highly similar ablation profiles despite their divergent cycle-probe accuracies, but more analysis would be needed to understand whether they have a similar learned computation.

5. Conclusion

Using permutations and their statistics as a testbed, we trained small transformers and applied a collection of interpretability tools to study how input notation affects learning. Because the underlying mathematical object is fixed and only the notation varies, we could isolate the effect of input notation on sample complexity and the learned computation.

This work provides a controlled setting for future mechanistic interpretability research. For example, circuit discovery methods could test whether learned circuits are stable across random seeds and hyperparameter choices, and whether this stability depends on input notation. Another direction is to examine how learned internal representations transfer across tasks, either through model stitching (15) or by finetuning a model trained on one permutation statistic to predict another. The discussion of task difficulty could also be complemented by a more formal analysis: for each (notation, task) pair one could ask whether the computation is expressible in C-RASP (36) (a programming language used to describe computations implementable by transformer-like architectures), and if so, what depth and program complexity are required. More broadly, this setup could be extended to other mathematical objects with multiple natural representations and statistics, such as graphs or Dyck paths. Such extensions could help identify the conditions under which different input notations lead models to learn similar internal mechanisms.

Acknowledgments

This work was conducted under the Laboratory Directed Research and Development Program at PNNL, a multi-program national laboratory operated by Battelle for the U.S. Department of Energy under contract DE-AC05-76RL01830.

References

[1] Yonatan Belinkov. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219, 2022.

[2] Yonatan Belinkov and Yonatan Bisk. Synthetic and natural noise both break neural machine translation. arXiv preprint arXiv:1711.02173, 2017.

[3] David Callan. Counting stabilized-interval-free permutations. *J. Integer Seq*, 7(1), 2004.

[4] François Charton. Linear algebra with transformers. *Trans. Mach. Learn. Res.*, 2022, 2021.

[5] Herman Chau, Helen Jenne, Davis Brown, Jesse He, Mark Raugas, Sara C. Billey, and Henry Kvinge. Machine learning meets algebraic combinatorics: A suite of datasets capturing research-level conjecturing ability in pure mathematics. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pages 7515–7554, 2025.

[6] Bilal Chughtai, Lawrence Chan, and Neel Nanda. A toy model of universality: Reverse engineering how networks learn group operations. In *International Conference on Machine Learning*, pages 6243–6267. PMLR, 2023.

[7] MohammadReza Davari, Stefan Horoi, Amine Natik, Guillaume Lajoie, Guy Wolf, and Eugene Belilovsky. Reliability of CKA as a similarity measure in deep learning. arXiv preprint arXiv:2210.16156, 2022.

[8] 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. *Transformer Circuits Thread*, 2021.

[9] Fabian Gröger, Shuo Wen, and Maria Brbić. Revisiting the platonic representation hypothesis: An aristotelian view, 2026.

[10] Michael Hanna, Sandro Pezzelle, and Yonatan Belinkov. Have faith in faithfulness: Going beyond circuit overlap when finding model mechanisms. arXiv preprint arXiv:2403.17806, 2024.

[11] Minyoung Huh, Brian Cheung, Tongzhou Wang, and Phillip Isola. Position: The platonic representation hypothesis. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pages 20617–20642. PMLR, 21–27 Jul 2024.

[12] Max Klabunde, Tobias Schumacher, Markus Strohmaier, and Florian Lemmerich. Similarity of neural network models: A survey of functional and representational measures. *ACM Computing Surveys*, 57(9):1–52, 2025.

[13] Donald Knuth. Permutations, matrices, and generalized young tableaux. *Pacific Journal of Mathematics*, 34(3):709–727, 1970.

[14] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. In *International Conference on Machine Learning*, pages 3519–3529. PMLR, 2019.

[15] Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 991–999, 2015.

[16] Ziming Liu, Eric Gan, and Max Tegmark. Seeing is believing: Brain-inspired modular training for mechanistic interpretability. *Entropy*, 26(1):41, 2023.

[17] Colin L Mallows. Patience sorting. *SIAM review*, 5(4):375, 1963.

[18] Sabrina J Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y Lee, Benoît Sagot, et al. Between words and characters: A brief history of open-vocabulary modeling and tokenization in NLP. arXiv preprint arXiv:2112.10508, 2021.

[19] Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability. *The Eleventh International Conference on Learning Representations*, 2023.

[20] Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of transformers with simple arithmetic tasks. arXiv preprint arXiv:2102.13019, 2021.

[21] Chris Olah, Nick Cammarata, Ludwig Schubert, Gabriel Goh, Michael Petrov, and Shan Carter. Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001, 2020.

[22] Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets. arXiv preprint arXiv:2201.02177, 2022.

[23] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219, 2019.

[24] Martin Rubey, Christian Stump, et al. FindStat - The combinatorial statistics database. http://www.FindStat.org.

[25] Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. How good is your tokenizer? On the monolingual performance of multilingual language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3118–3135, 2021.

[26] Peter H Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966.

[27] Jie Shen and Christos A Nicolaou. Molecular property prediction: recent trends in the era of artificial intelligence. *Drug Discovery Today: Technologies*, 32:29–36, 2019.

[28] Dashiell Stander, Qinan Yu, Honglu Fan, and Stella Biderman. Grokking group multiplication with cosets. arXiv preprint arXiv:2312.06581, 2023.

[29] Richard P Stanley. *Enumerative combinatorics, volume 1*. Cambridge Studies in Advanced Mathematics, 2011.

[30] Aaquib Syed, Can Rager, and Arthur Conmy. Attribution patching outperforms automated circuit discovery. In *Proceedings of the 7th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pages 407–416, Miami, Florida, US, November 2024. Association for Computational Linguistics.

[31] The Sage Developers. *Sage Reference Manual: Combinatorics*. The Sage Development Team, version 10.6 edition, 2024. URL https://doc.sagemath.org/html/en/reference/combinat/sage/combinat/permutation.html.

[32] Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. Investigating gender bias in language models using causal mediation analysis. In *Advances in Neural Information Processing Systems*, volume 33, pages 12388–12401, 2020.

[33] 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. arXiv preprint arXiv:2211.00593, 2022.

[34] Phil Wang. x-transformers: A concise but complete full-attention transformer. https://github.com/lucidrains/x-transformers, 2020.

[35] Wilson Wu, Louis Jaburi, Jacob Drori, and Jason Gross. Unifying and verifying mechanistic interpretations: A case study with group operations. arXiv preprint arXiv:2410.07476, 2024.

[36] Andy Yang and David Chiang. Counting like transformers: Compiling temporal counting logic into softmax transformers. arXiv preprint arXiv:2404.04393, 2024.

[37] Lin Zhang, Wenshuo Dong, Zhuoran Zhang, Shu Yang, Lijie Hu, Ninghao Liu, Pan Zhou, and Di Wang. EAP-GP: Mitigating saturation effect in gradient-based automated circuit identification. *Advances in Neural Information Processing Systems*, 38: 141899–141925, 2026.

[38] Yi Zhang, Arturs Backurs, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, and Tal Wagner. Unveiling transformers with LEGO: a synthetic reasoning task. arXiv preprint arXiv:2206.04301, 2022.

[39] Ziqian Zhong, Ziming Liu, Max Tegmark, and Jacob Andreas. The clock and the pizza: Two stories in mechanistic explanation of neural networks. *Advances in Neural Information Processing Systems*, 36, 2024.

[40] Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Joshua Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? A study in length generalization. In *International Conference on Learning Representations*, volume 2024, pages 15898–15926, 2024.

Appendix A. Related work

A.1. Universality of representations

Related to our work is the study of universality of representations. The *platonic representation hypothesis* posits that as models scale, representations converge, even across modalities (11). This hypothesis is debated: Gröger et al. (9), for example, points out that some similarity measures are inflated by increasing model depth and width, and proposes ways to calibrate these scores. We ask a more narrow question: do models converge to the same hidden representation when given different input representations of the same data?

A.2. Learning on permutation data

Permutation composition has been used to study grokking (22) and in mechanistic interpretability studies (38; 16; 6; 28; 35). Chau et al. (5) trained small transformer models to predict *Schubert structure constants* from permutation data, a task motivated by an open problem in algebraic combinatorics. In contrast, we train models to predict known permutation statistics, where our focus is on varying the notation used to encode the underlying permutation.

A.3. Mechanistic interpretability

Mechanistic interpretability aims to understand ML models by reverse engineering the algorithms they learn. A central goal in mechanistic interpretability is *circuit discovery*: identifying a sparse subnetwork, or circuit, of important model components and their interactions. A variety of tools aim to identify and describe these small subgraphs which preserve some target behavior, notably inspection of activations (21), causal intervention and (edge) attribution patching (10; 30; 32; 37). Our goal is not to recover complete circuits for each (notation, task) combination. Instead, we use head ablation and linear probing as lightweight circuit-analysis tools to compare where and in what form task-relevant information appears when models are trained on different input notations.

Appendix B. Permutation representations and statistics

B.1. Converting from inversion vector to one-line notation

Given the inversion vector $(v_1, v_2, \ldots, v_n)$ notation that represents a permutation in $\mathcal{S}_n$, convert to one-line notation as follows: for $i = 1, 2, \ldots, n$, place i so that exactly v_i larger values precede it.

B.2. Converting from cycle notation to one-line notation

For each $i = 1, 2, \ldots, n$ find i in the cycle notation $(a_1 \cdots a_k)(b_1 \cdots b_\ell) \cdots$. If $i = a_m$, then $\sigma_i = a_{m+1}$, or a_1 if a_m is the last element of the cycle.

B.3. Computing the length of the longest increasing subsequence from one-line notation

The length of the longest increasing subsequence can be computed from one-line notation in a few different ways, including but not limited to dynamic programming, converting to Standard Young Tableaux via the Robinson-Schensted-Knuth algorithm (13), and using *patience sorting*, which was originally introduced as a card sorting algorithm (17).

Given a shuffled deck of cards $\sigma = c_1 c_2 \cdots c_n \in \mathcal{S}_n$, patience sorting has two steps: (1) Form a new pile r_1 using the card c_1. (2) For each $i = 2, \ldots, n$ consider the cards $d_1, d_2, \ldots, d_k$ on top of the existing piles $(r_1, \ldots, r_k)$. If c_i is greater than all of the top cards, start a new pile. Otherwise, place it on top of the leftmost pile whose top card is larger. When complete, the number of piles left will be the length of the longest increasing subsequence.

Appendix C. Methods

C.1. Model architecture and hyperparameters

Tokenizer While one-line notation and inversion vector notation produce sequences of the same length (N), permutations written in cycle notation range from length 2 (the identity, encoded as `( )`) to $N + 2 * \lfloor \frac{N}{2} \rfloor$. We pad shorter sequences with `[PAD]` tokens to the maximum length observed in the dataset, and construct a boolean attention mask that marks padding positions as invalid.

Positional embedding We use scaled sinusoidal embeddings for the positional embedding.

Hyperparameters All models were trained using the AdamW optimizer with learning rate 3×10^{-4}, weight decay 0.01, and batch size 128. Models were trained for 200 epochs, with the exception of the model training for the sample complexity results in Appendix D.1, where the numbers of epochs was adjusted so that each model had approximately the same number of updates.

C.2. Representational similarity

C.2.1. CKA

We computed Linear CKA as follows:

$$\text{CKA}(X, Y) = \frac{||Y_c^T X_c||_F^2}{||Xc^T Xc||_F ||Yc^T Yc||_F}$$

where X_c, Y_c are the column-centered feature matrices.

C.2.2. Procrustes

As in the linear probing experiments, we split the test set into a Procrustes set and a held-out evaluation set. On the fitting set with means μ_X, μ_Y, we solve

$$(Q^*, \alpha^*) = \underset{Q^\top Q = I,\ \alpha \in \mathbb{R}}{\arg\min} \|\alpha(X_{\text{fit}} - \mu_X)Q - (Y_{\text{fit}} - \mu_Y)\|_F^2 .$$

We compute this using the singular value decomposition of the cross-covariance matrix between the centered fitting activations. We then compute the relative Frobenius reconstruction error,

$$E_{X \to Y} = \frac{\|\alpha^*(X_{\text{eval}} - \mu_X)Q^* - (Y_{\text{eval}} - \mu_Y)\|_F}{\|Y_{\text{eval}} - \mu_Y\|_F},$$

so lower values indicate more similar representations.

C.3. Attention head ablation

Each experiment removes a single head's contribution to the residual stream while leaving all other components unchanged. A single head (l, h) is ablated by zeroing the columns of the output projection matrix $W_O^{(l)}$ that correspond to head h. If each head occupies $d_{\text{head}} = d_{\text{model}}/H$ input channels of $W_O^{(l)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$, we set

$$W_O^{(l)}[:,\ h \cdot d_{\text{head}} : (h+1) \cdot d_{\text{head}}] = \mathbf{0}$$

during the forward pass, then restore the original weights afterward.

We define the ablation effect as $\Delta_{\text{acc}}^{(l,h)} = \text{acc}_{\text{baseline}} - \text{acc}_{\text{ablated}}^{(l,h)}$.

C.4. OOD generalization for cycle-notation models

Our main dataset uses the common convention of starting each cycle with the smallest number in that cycle, and then ordering cycles based on their starting element, in increasing order from left to right. But the numbers within a cycle can be cyclically permuted and the cycles can be permuted without changing the underlying cycle. We evaluate our trained models on (i) cycles where the numbers within the cycle have been cyclically rotated to the right by 1 position (ii) cycles rotated so that their largest element is first (iii) cycles that have been randomly rotated.

Appendix D. Additional results

Table 1 summarizes the results of training 4-layer, 4-head transformers as described in Appendix C on each (notation, task) pair.

D.1. Sample complexity

In the experiments that created Figure 1, we train each model on a subset of $\mathcal{S}_8$, increasing the size of training data on a logarithmic scale while keeping the validation set and test set constant. We vary the number of epochs so that the total number of training steps is approximately fixed.

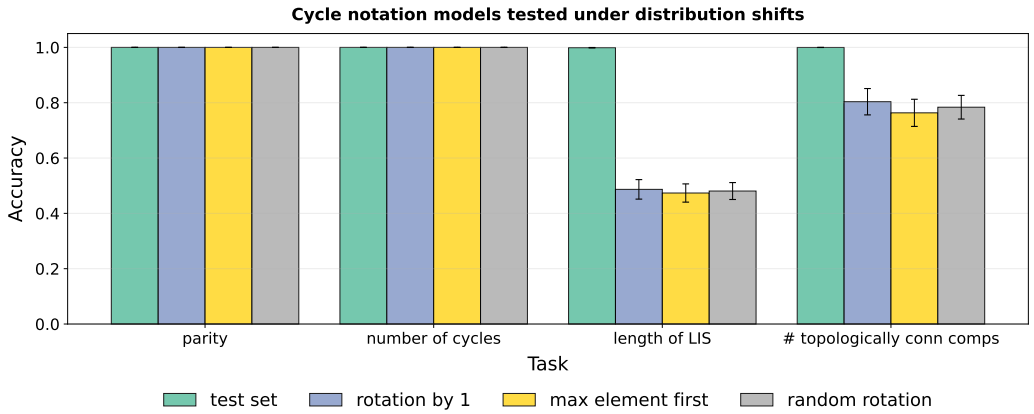

Figure 4: Accuracy of models trained on cycle notation under various distribution shifts.

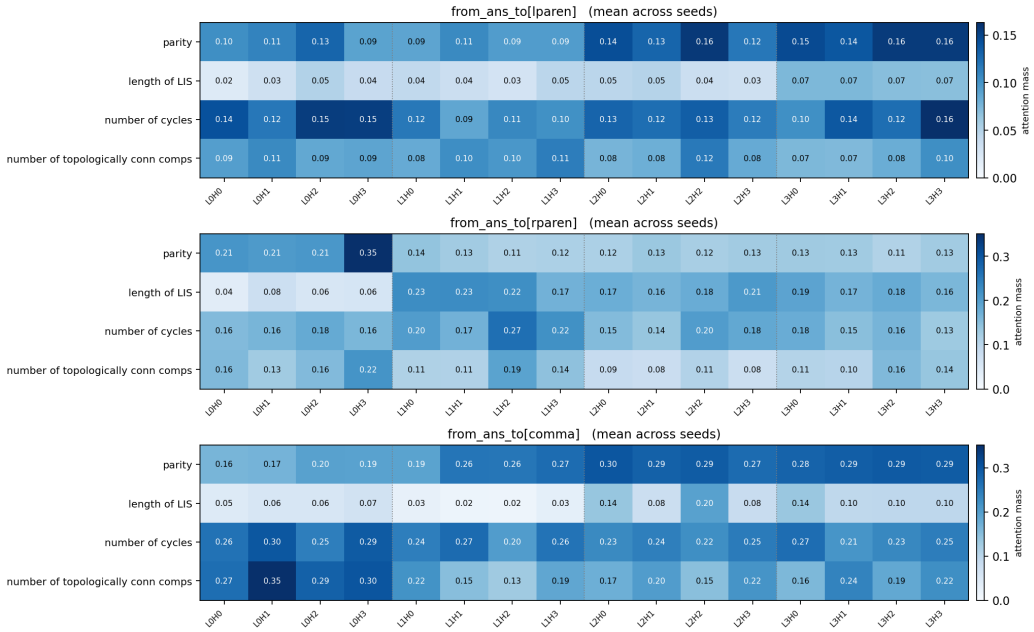

Figure 5: Per head attention from answer position to different structural tokens for models trained on cycle notation.

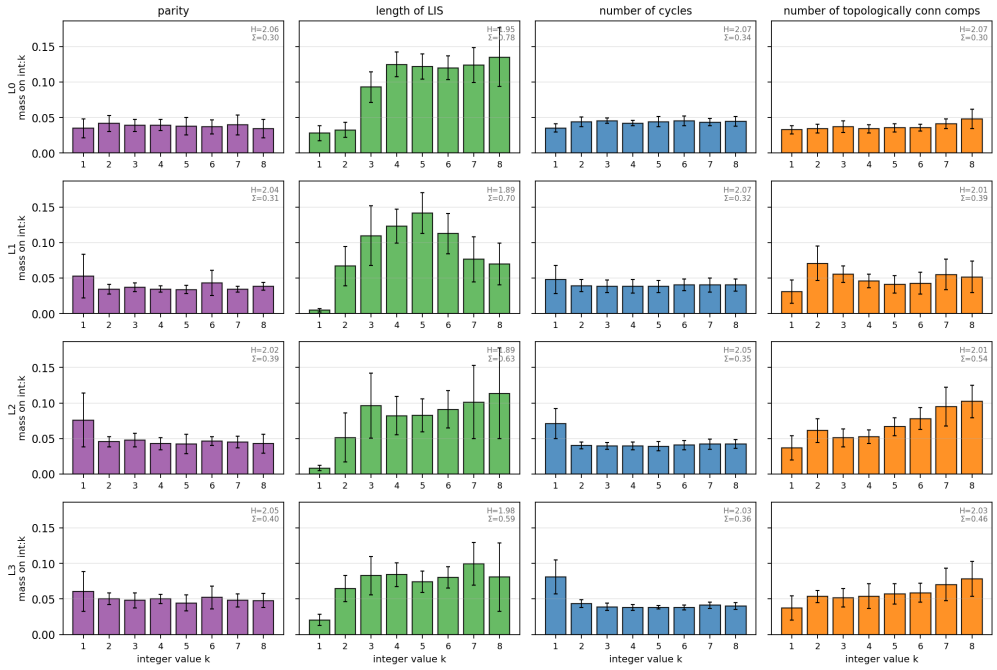

Figure 6: Per head attention from answer position to different integer tokens for models trained on cycle notation.

Task	one-line	cycle notation	inversion vector
parity	0.497 ± 0.000	1.000 ± 0.000	0.999 ± 0.003
number of cycles	1.000 ± 0.000	1.000 ± 0.000	0.991 ± 0.004
length of LIS	1.000 ± 0.000	0.999 ± 0.001	1.000 ± 0.000
# topologically conn comps	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000

Table 1: Test accuracy ($d_{\text{model}} = 128$, 4 heads, 4 layers) transformers; mean $\pm$ one standard deviation across 10 seeds per (task, representation) cell. Full-data runs only.

D.2. OOD generalization results

We hypothesize that the models trained on cycle notation primarily rely on structural tokens for the parity and number of cycles tasks. Figure 4 shows the results from testing on OOD data. We see that performance on the parity and number of cycles task is not affected by the convention chosen for cycle notation. This is expected, since the parity of a permutation is the parity of the number of even-length cycles and the number of cycles is the number of left or right parentheses in the cycle. The attention patterns support the hypothesis that the models rely primarily on the structural tokens. For the number of cycles task, we see that the model uses the cycle structure tokens more than the integer tokens (see Figure 5) and that attention from ANS is nearly uniform across tokens 1-8 (see Figure 6).

D.3. Attention head ablation

Ablation drop profiles by representation are plotted in Figure 7. Spearman correlation of ablation profiles are shown in Figure 8. The bootstrapped standard error is reported for each, computed by resampling model seeds with replacement and recomputing each notations mean ablation vector and ρ, 500 times. There are no within-notation results reported for one-line notation on parity task because the model failed to learn, so the correlation of ablation profiles is not well defined.

D.4. Additional linear probe results

Figures 9 and 10 show additional linear probe results for one-line notation. Figures 11 and 12 show additional linear probe results for characteristics of cycle notation.

D.5. Representation similarity results

Figures 13–16 show the CKA results for the four tasks. Each plot shows three heatmaps, one for each pair of input notations. Figures 17–20 show the orthogonal Procrustes results for the four tasks.

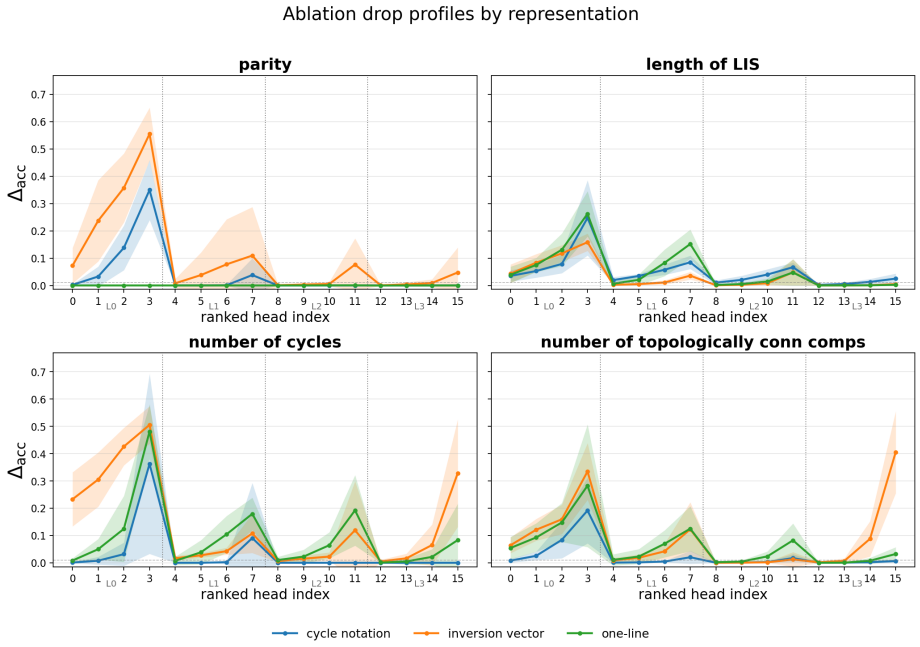

Figure 7: For each task, the mean accuracy drops as a function of the layer index. Within each layer, heads are sorted so the head with smallest drop has the lowest rank index, to account for the fact that heads within each layer are arbitrarily labeled. Shaded regions show ±1 standard deviation across 10 seeds.

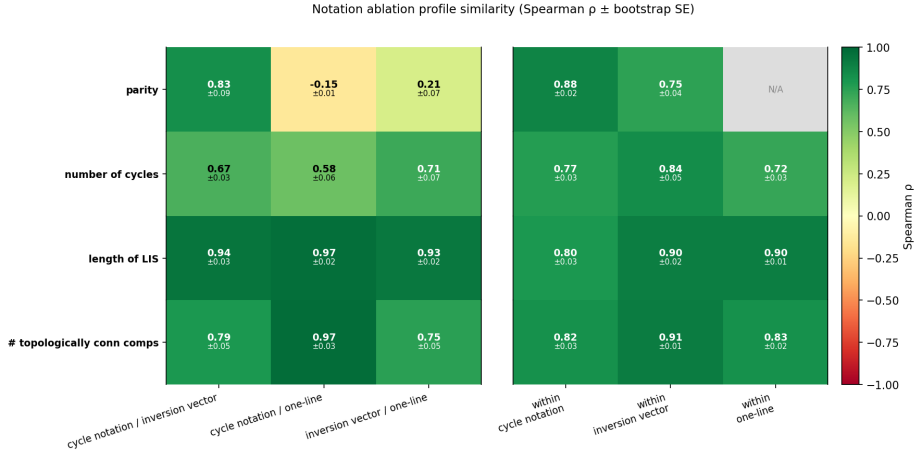

Figure 8: Spearman correlation of pairs of notations (left) and within notations (right) showing the similarity of ablation profiles on the various tasks.

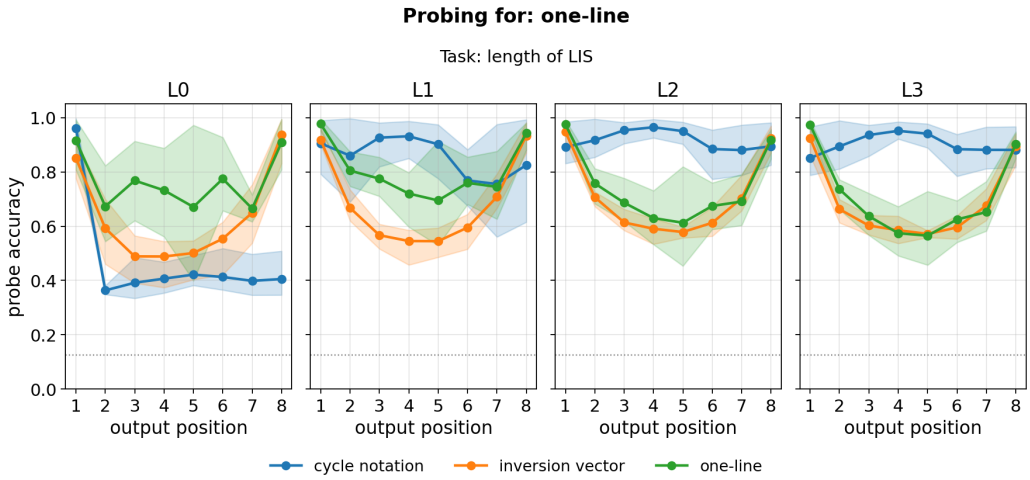

Figure 9: Linear probe with target one-line notation for models trained on the length of the LIS task, as shown in Figure 3. This figure shows the accuracy by token.

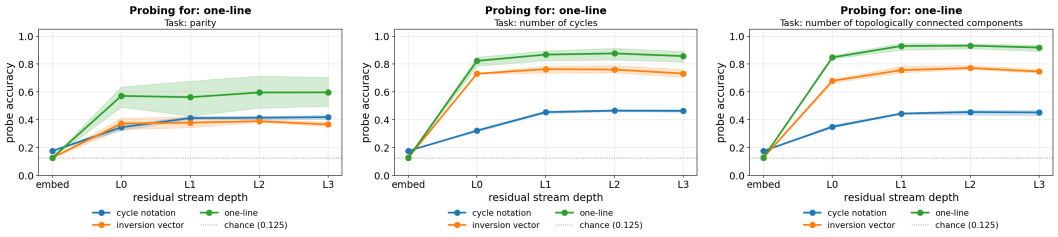

Figure 10: Linear probe with target one-line notation for models trained to learn parity, the number of cycles, and the number of topologically connected components.

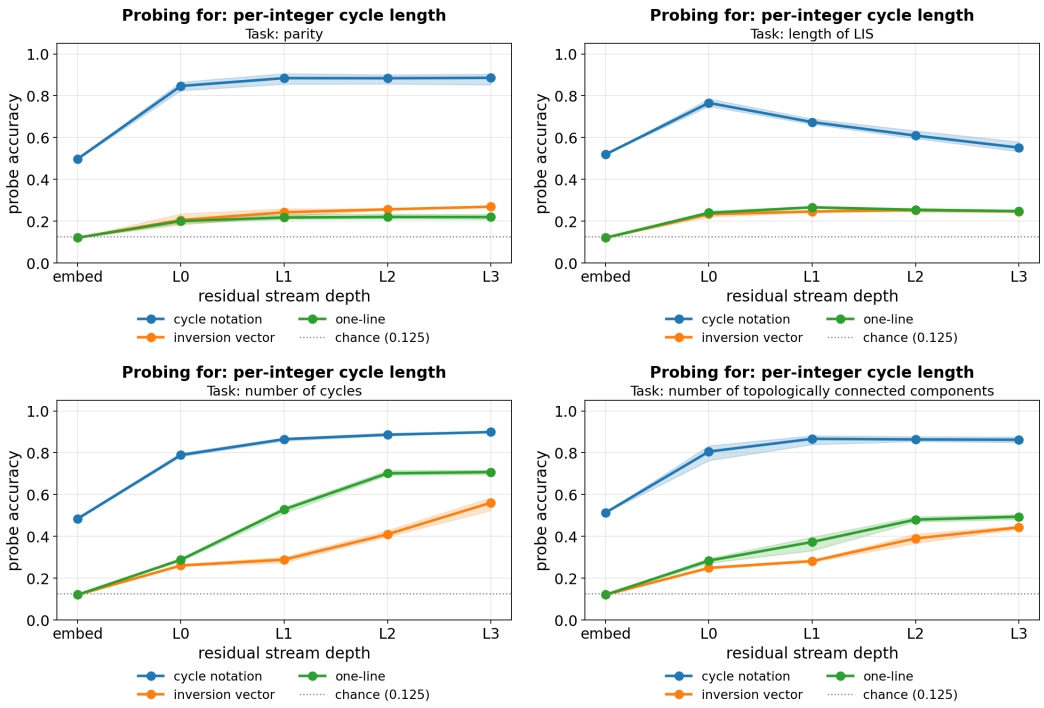

Figure 11: Linear probe with target per-integer cycle length, that is, the vector where $v(i)$ gives the length of the cycle containing i. Upper row: parity (left), length of LIS (right). Lower row: number of cycles (left), number topologically connected components (right).

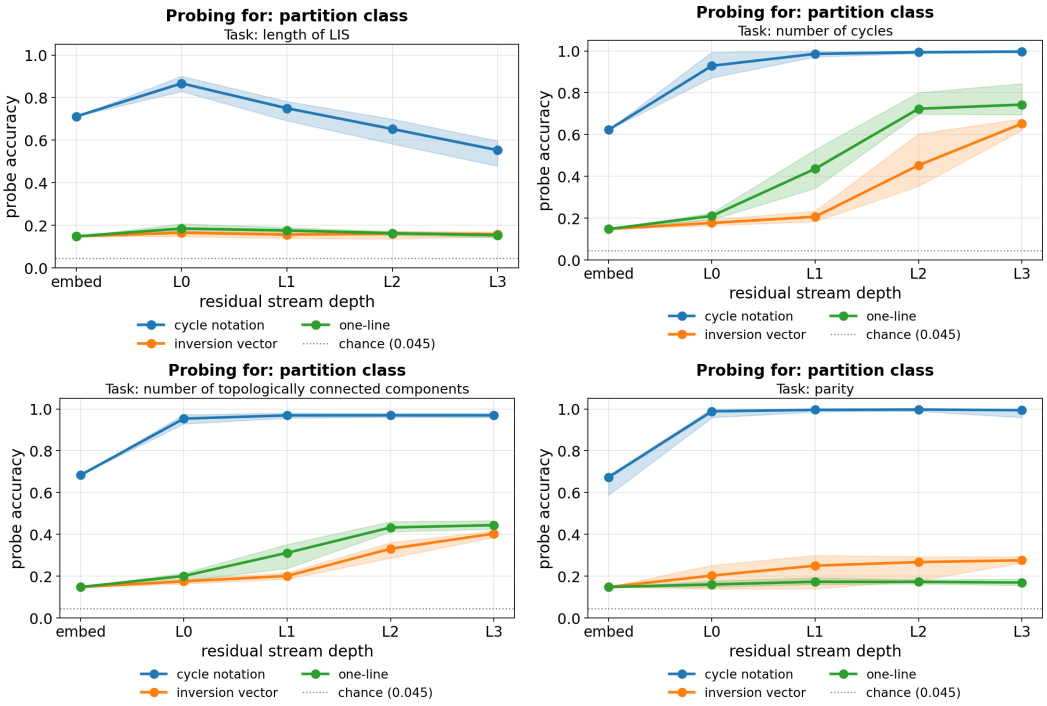

Figure 12: Partition class linear probe accuracy for the four tasks.

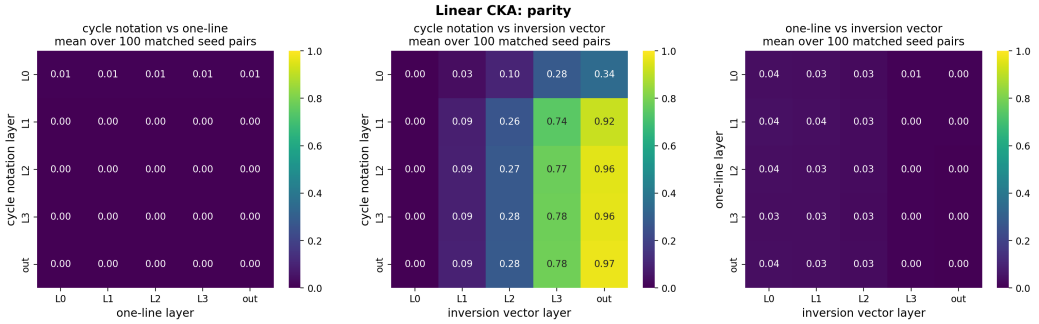

Figure 13: Cross-representational CKA for the parity task.

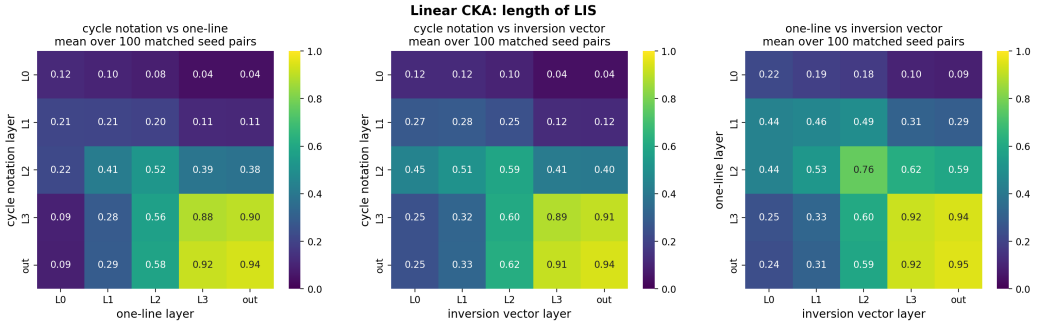

Figure 14: Cross-representational CKA for the longest increasing subsequence length task.

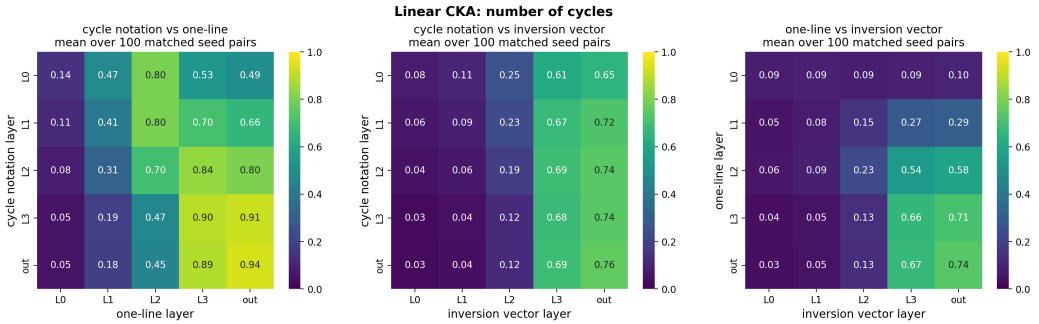

Figure 15: Cross-representational linear CKA for the number of cycles task.

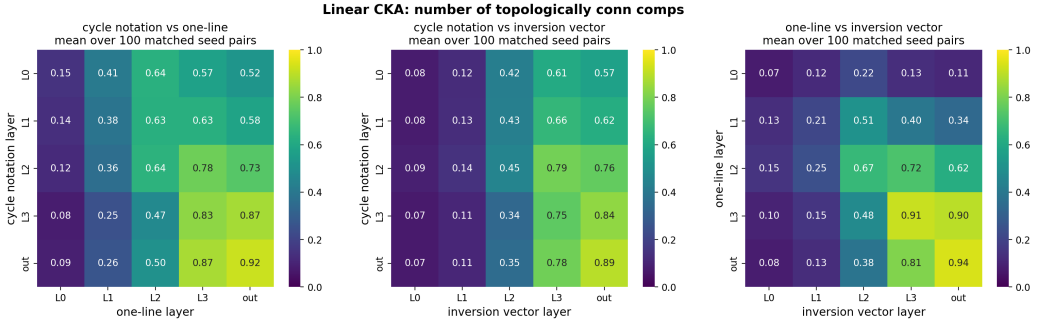

Figure 16: Cross-representational CKA for the number of topologically connected components task.

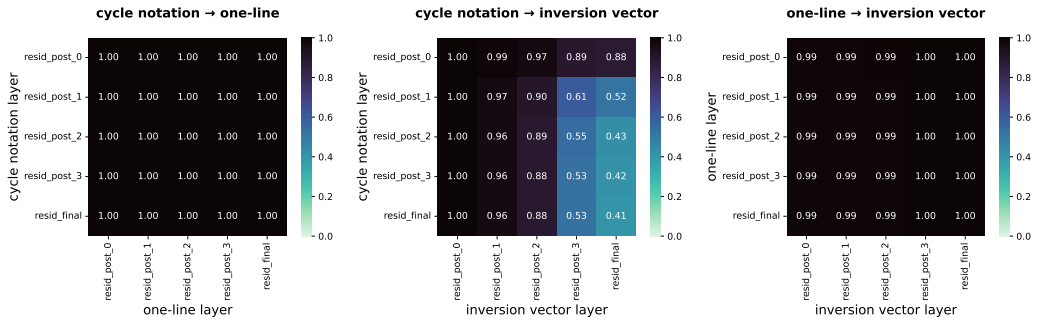

Figure 17: Cross-representational Procrustes for the parity task. The plot shows the relative error on the holdout test set.

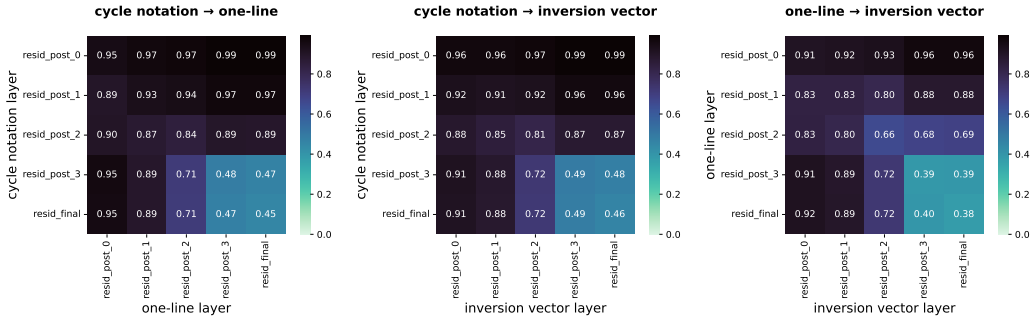

Figure 18: Cross-representational Procrustes for the longest increasing subsequence length task. The plot shows the relative error on the holdout test set.

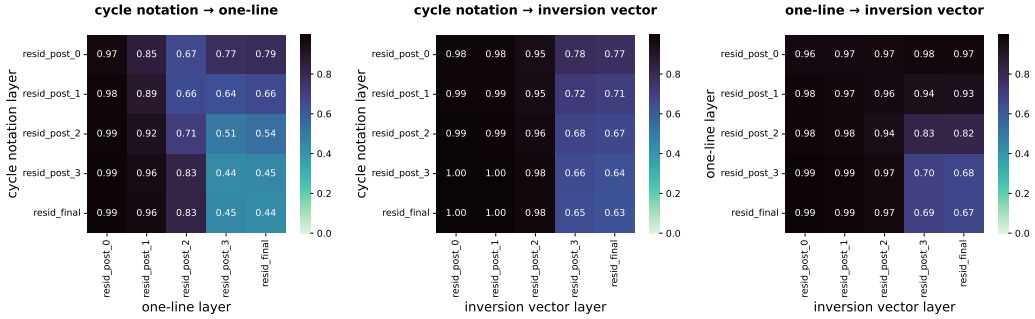

Figure 19: Cross-representational Procrustes for the number of cycles task. The plot shows the relative error on the holdout test set.

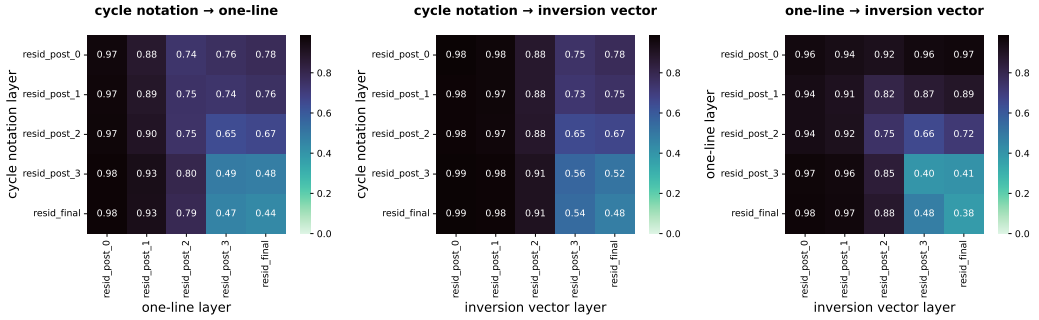

Figure 20: Cross-representational Procrustes for the number of topologically connected components task. The plot shows the relative error on the holdout test set.