Permutations as a testbed for studying the effect of input representations on learning

Sarah Scullen [1] **Davis Brown** [1 2] **Robert Jasper** [1] **Henry Kvinge** [1 3] **Helen Jenne** [1]

Abstract

Quality data is crucial for deep learning. However, relative to progress in model training and data curation, there is a lesser focus on understanding the effects of how data is encoded and passed to the neural network— the "data representation." This is especially true for non-textual domains, where there are often challenges in distinguishing between the difficulty of the learning task versus difficulties arising merely from the format of the input data. We propose using permutations, which have multiple natural mathematical representations, as a systematic way to study how task difficulty and learning outcomes are influenced by the choice of input data representation. In this setting, we find that the model performance on a data representation can change significantly with the number of examples and architecture type; however, with enough examples most tasks are learned regardless of data representation.

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 a model's learning outcomes. In natural language processing, the transition from bag of words encoding to context-aware embeddings demonstrates how representation choice can dramatically influence downstream model performance. Similarly, in a variety of other domains – from molecular property prediction (Shen & Nicolaou, 2019), to audio signal processing (Purwins et al., 2019), to intrusion detection in cybersecurity (Arnaldo et al., 2017) – the choice of representation can significantly affect model behavior. Despite these observations, we lack systematic methods for understanding how representation choice affects task difficulty, model performance, and the algorithms that models discover.

We propose using permutations and permutation statistics to study how input representation affects learning. Permutations provide a setting in which there are many ways to naturally represent the data, which each emphasize different structural properties. Additionally, there are a wide range of statistics of interest that can be computed from permutations, with the FindStat database (Rubey et al.) having over 400 statistics that can be computed from a permutation, providing many avenues for potential classification problems. Studying the pairwise combinations of representations and tasks on the same underlying mathematical objects allows one to isolate the effect of representation choice on learning. This controlled setting enables the study of fundamental questions:

- Learned algorithms: How does the input encoding affect the algorithm the model learns?
- Interpretability: Do certain representations consistently yield more interpretable models?
- Task complexity: Which tasks are challenging regardless of input representation or model architecture?
- Model selection: Under which task-data pairings do simpler models outperform complex ones?
- Learning speed: How does representation choice affect the number of examples needed to learn?

In this work, we begin to study these questions, training small MLPs and transformers on a variety of representation-task combinations. Our key contributions include:

[*]Equal contribution [1]Pacific Northwest National Laboratory [2]University of Pennsylvania [3]University of Washington. Correspondence to: Helen Jenne <helen.jenne@pnnl.gov>.

Accepted at Methods and Opportunities at Small Scale (MOSS), ICML 2025, Vancouver, Canada.

- Providing a framework combining five permutation representations with twelve classification tasks, where each task has a known algorithm in at least one representation.
- Finding that when there are fewer examples (permutations on smaller values of N) certain representations provide larger advantages for learning, but once N is sufficiently large, nine of the twelve tasks are learned using all representations.
- Enabling future mechanistic interpretablility research into how representation choice influences the internal algorithms models discover.

2. Related work

2.1. Sparse-parity learning

Work on sparse-parity learning (Edelman et al., 2023; Morwani et al., 2024) modulates model width, dataset size, and training time, finding that models rarely learn a XOR bit unless extreme width supplies a "lottery-ticket" neuron or training gradually builds a Fourier/group basis to expose it. Our experiments examine a related domain: permutation parity is difficult to learn across one-line, matrix, reduced-word—and even inversion-vector or Lehmer cocode—encodings, indicating that merely surfacing the linear bit is not enough. Whereas (Edelman et al., 2023; Morwani et al., 2024) fix the Boolean input and vary model capacity or training dynamics, we hold those factors constant and instead vary natural permutation representations.

2.2. Learning with permutation data and mechanistic interpretability

The task of permutation composition has been used in research on grokking (Power et al., 2022) and in interpretability work (Zhang et al., 2022; Liu et al., 2023; Chughtai et al., 2023; Stander et al., 2023; Wu et al., 2024). Much of this research falls within mechanistic interpretability, which aims to understand ML models by reverse engineering the algorithms they learn.

While modular arithmetic has become a classic testbed for mechanistic interpretability (Nanda et al., 2023; Zhong et al., 2024; Yip et al., 2024), it is just one example of group composition on a finite group – specifically, the cyclic group. Given this foundation, studying permutation composition (equivalently, group composition in permutation groups) is a logical next step. For example, (Chughtai et al., 2023) trained MLP and transformer models to perform group composition on seven finite groups, including the permutation groups S_5 and S_6, and aimed to reverse engineer how the networks learned these operations. They observed that the learned features correspond to the group's irreducible representations. However, competing explanations have emerged: Stander et al. found that in the same experimental setup with S_5 and S_6, the model used the coset structure of the group to perform composition (Stander et al., 2023). Gross et al. offered a third mechanistic explanation based on group-theoretic structure that both previous studies had overlooked (Wu et al., 2024).

Our work provides a testbed where mechanistic interpretability studies can include the effect of different input encodings on learned algorithms, the difficulty of the tasks and the relationship between representations, task difficulty, and ease of interpretability.

3. Experimental design

Our experimental design enables systematic study of how representation choice affects task difficulty and learning outcomes. We hold underlying mathematical objects, tasks, and model architecture constant while varying only the input representation. We selected twelve classification tasks that capture different aspects of permutation structure, including both local properties like *peaks* and *descents*, and global properties like *order*. We chose five different permutation representations that emphasize different permutation characteristics. Most tasks have at least one simple algorithm for at least one representation, allowing one to check whether the models learn these solutions.

In this section, we first briefly introduce permutations and definitions used throughout this paper. A thorough introduction to permutations and permutation statistics can be found in (Stanley, 2011). We next describe the tasks and representations that we trained small MLPs and transformers on. Details of model training are left to the Appendix.

3.1. Permutations: key concepts

A permutation $\sigma \in \mathcal{S}_n$ is a bijective function from the set $\{1, 2, \ldots, n\}$ to itself. The one-line representation of a permutations (often referred to as *one-line notation*) is the vector $(\sigma(1), \sigma(2), \ldots, \sigma(n))$. For example, the permutation that maps $1 \mapsto 2$,

$2 \mapsto 5$, $3 \mapsto 3$, and $4 \mapsto 4$ and $5 \to 1$ has one-line representation 25341. Other fundamental concepts we need include:

- **Inversions**: Pairs (i, j) where $i < j$ but $\sigma(i) > \sigma(j)$. In the permutation 25341, $(2, 3)$ is one of six inversions.
- **Descents**: Positions i where $\sigma(i) > \sigma(i + 1)$. The descents in the permutation 25341 are 2 and 4.
- **Cycles**: Each permutation can be written as a product of disjoint *cycles*. In this representation, every integer in the cycle is mapped to the next integer in the cycle. The permutation 25341 is written using its cycle representation as $(125)(3)(4)$.
- **Reduced words:** Permutations can be written (non-uniquely) as sequences of transpositions of adjacent elements. For example, the permutation 25341 can be written in cycles as $(1, 2)(2, 3)(3, 4)(4, 5)(3, 4)(2, 3)$, where the cycles are read from right to left. Abbreviating the adjacent transposition $(i, i + 1)$ as s_i, we say that $s_1 s_2 s_3 s_4 s_3 s_2$ is a *reduced decomposition* of the permutation 25341, and the 123432 is a *reduced word* of 25341.

3.2. Tasks

We study twelve classical permutation statistics, selected to have natural algorithms in different representations. More information about these statistics and examples can be found in the FindStat database (Rubey et al.) and the Sage documentation (The Sage Developers, 2024).

- **Length**: the number of inversions in the permutation or, equivalently, the length of a reduced word for the permutation (also called Coxeter length).
- **Parity**: Whether the permutation has even or odd (Coxeter) length.
- **Major index**: The sum of all of the descents.
- **Number of descents**: Count of positions i where $\sigma(i) > \sigma(i + 1)$.
- **Number of fixed points**: Count of values i where $\sigma(i) = i$.
- **Number of peaks**: Count of positions i where $\sigma(i - 1) < \sigma(i)$ and $\sigma(i) > \sigma(i + 1)$.
- **Longest increasing subsequence length**: Maximum length of $i_1 < i_2 < \cdots < i_k$ with $\sigma(i_1) < \sigma(i_2) < \cdots < \sigma(i_k)$.
- **Number of nestings** Count of pairs (i, j) that form nestings in the permutation's arc diagram: either $j < i < \sigma(i) < \sigma(j)$ or $\sigma(j) < \sigma(i) \leq i < j$ (Corteel, 2007).
- **Order**: The smallest positive integer m so that $\sigma^m = id$. This is the least common multiple of the lengths of its cycles.
- **Number of cycles**: The count of cycles in the disjoint cycle decomposition of the permutation (including cycles of length 1).
- **Support cardinality**: The number of distinct indices that appear in a reduced word of the permutation. Note that this is independent of the choice of reduced word.
- **Number of stack sorts**: The minimum number of stack sorts needed to sort a permutation. Details on *stack sorting* can be found in the Appendix.

3.3. Representations

The difficulty of the tasks in the previous section depends on their representation. We compare five representations that encode different structural aspects of permutations:

- **Permutation matrix**: The $n \times n$ matrix where entry $(i, j) = 1$ if $\sigma(i) = j$, and 0 otherwise. This is the one-hot encoding of the one-line representation.
- **Inversion vector**: The vector where the entry i counts how many $j > i$ appear before i in the permutation. For the permutation 25341, the inversion vector is $(4, 0, 1, 1, 0)$.
- **Lehmer cocode**: The vector where entry i counts how many $j < i$ have $\sigma(j) > \sigma(i)$. The Lehmer cocode of the permutation 25341 is $(0, 0, 1, 1, 4)$. Note that applying σ to the Lehmer cocode gives the inversion vector.
- **Major code**: The vector $(m_1 - m_2, m_2 - m_3, \ldots, m_n)$, where m_i is the sum of all of the descents of the permutation obtained by erasing letters smaller than i from the permutation. For the permutation 25341, $m_1 = 6$, $m_2 = 2$, $m_3 = 2$ and $m_4 = m_5 = 0$, so the major code is $(4, 1, 0, 1, 0)$.
- **Lexicographically minimal reduced word**: The reduced word for the permutation that is smallest in lexicographic order. Since different permutations may have reduced words of different length, we pad the reduced words with zeros so that they are all the same length.

3.4. Representation-task interactions

Certain statistics can be easily computed from certain representations:

- **Length and parity**: The length can be obtained by taking the sum of the inversion vector or Lehmer cocode, and the number of nonzero entries of the lexicographically minimal reduced word. The parity is the parity of the length.
- **Fixed points**: The number of fixed points is the trace of the permutation matrix.
- **Descents**: The number of descents is exactly the number of ascents in the Lehmer cocode.
- **Major index:** The major index is the sum of the major code.
- **Support cardinality:** The support cardinality is the number of distinct integers that appear in the reduced word.

This is summarized in Table 1.

	Parity	Length	Long. inc sseq	Major Index	# Cycles	# Descents	# Fixed Points	# Nestings	# Peaks	# Stack Sorts	Order	Support Card.
Inversion Vector	○	●										
Lehmer Cocode	○	●				○						
Major Code				●								
Permutation Matrix				○		○	●		○			
Lexmin Reduced Word	○	●										●

Table 1. Task-representation pairs: ● = directly encoded (e.g., taking the sum or computing the length of a list), ○ = simple algorithm (e.g., comparing adjacent elements).

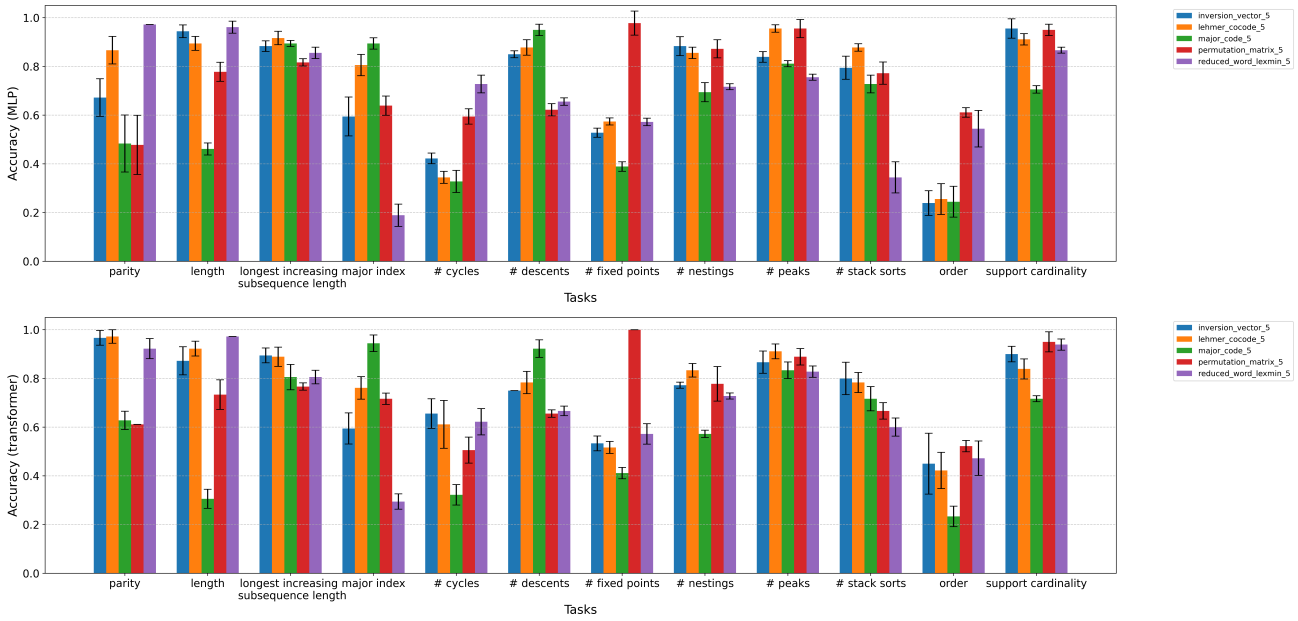

Figure 1. Accuracy averaged over five training runs of classifiers (MLPs in the top plot and transformers in the bottom plot), for each pair of permutation representation and task. Standard deviation is shown by error bar. Permutations from S_5.

4. Results

Once sufficient data was available ($N = 8$), all models achieved either very high or perfect accuracy on nine out of the twelve tasks regardless of representation, as shown in Figure 2. The three exceptions were the parity, order, and cycle counting tasks. MLPs acheived perfect test accuracy with at least one of the representations with sufficient data, while the

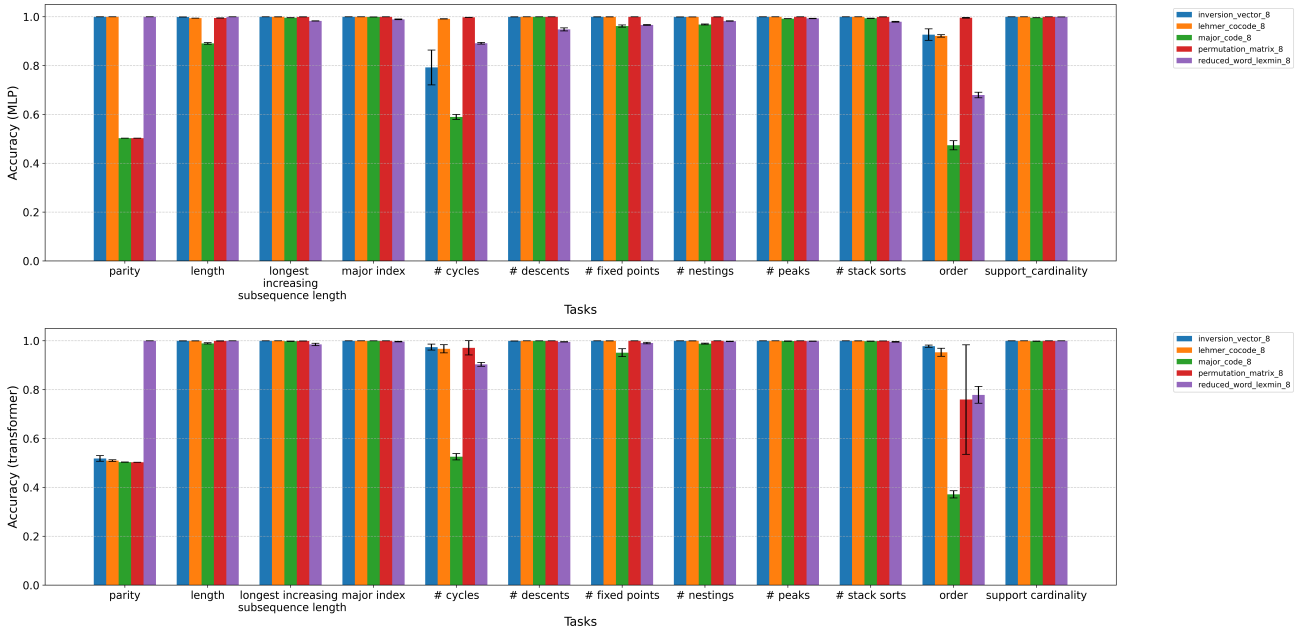

Figure 2. Accuracy averaged over five training runs of classifiers (MLPs in the top plot and transformers in the bottom plot), for each pair of permutation representation and task. Standard deviation is shown by error bar. Permutations from S_8.

transformers achieved high accuracy on every task with at least one of the representations, but notably failed to reach perfect accuracy on the order or counting cycles tasks.

The amount of data needed to achieve strong performance varied with the representation choice. For example, learning to compute the major index or the number of fixed points from the reduced word required much larger N than with the other representations. MLPs demonstrated superior data efficiency on specific representation-task pairs, requiring smaller N to achieve high performance on fixed points (with inversion vector or Lehmer cocode representations), major index (with the reduced word representation), and length and cycle counting (with the permutation matrix representation).

We also observed unexpected task-specific phenomena that challenged our intuition about task difficulty. Most surprisingly, computing the parity was much harder than computing length. The transformer models achieved 100% accuracy on the length task with every representation at $N = 8$, but performed similar to chance level on parity. Figure 1 shows performance results that vary in alignment with the expected relative difficulty of different tasks (see Table 1).

5. Discussion and conclusion

Our study into permutation representations underscores the impact that data representation choice can have on learning outcomes. This research provides insights into how permutations, with their diverse representation options, serve as an ideal framework for examining the influence of representation on model performance. Our findings suggest that permutations, with their inherent flexibility in representation and the rich variety of associated statistics, allow for systematic analysis of task complexity relative to the choice of input representation. We highlight the ability to isolate the effects of representation choice on learning through controlled experiments, thus enabling exploration into fundamental questions regarding learned algorithms, interpretability, task complexity, model selection, and learning speed. Our results indicate that certain representations have advantages when dealing with limited data (e.g. smaller permutations), while for larger datasets, more tasks are reliably learned across all representations.

This study sets the stage for further research on how representation choice affects the internal algorithms that models discover. Specifically, to further understand the relationship between data representation and learning outcomes, in future work we plan to compare models' internal representations across the different data representations, for different data representations, tasks, and values of N. Additionally, error analysis experiments would provide a more fine-grained understanding, beyond accuracy, of how learning is affected by data representation. Future work should also include extending this approach to different data modalities beyond permutations.

References

Arnaldo, I., Cuesta-Infante, A., Arun, A., Lam, M., Bassias, C., and Veeramachaneni, K. Learning representations for log data in cybersecurity. In *Cyber Security Cryptography and Machine Learning: First International Conference, CSCML 2017, Beer-Sheva, Israel, June 29-30, 2017, Proceedings 1*, pp. 250–268. Springer, 2017.

Chughtai, B., Chan, L., and Nanda, N. A toy model of universality: Reverse engineering how networks learn group operations. In *International Conference on Machine Learning*, pp. 6243–6267. PMLR, 2023.

Corteel, S. Crossings and alignments of permutations. *Advances in Applied Mathematics*, 38(2):149–163, 2007.

Edelman, B. L., Goel, S., Kakade, S., Malach, E., and Zhang, C. Pareto frontiers in neural feature learning: Data, compute, width, and luck. *arXiv preprint arXiv:2309.03800*, 2023.

Liu, Z., Gan, E., and Tegmark, M. Seeing is believing: Brain-inspired modular training for mechanistic interpretability. *Entropy*, 26(1):41, 2023.

Morwani, D., Edelman, B. L., Oncescu, C.-A., Zhao, R., and Kakade, S. Feature emergence via margin maximization: Case studies in algebraic tasks. *arXiv preprint arXiv:2311.07568*, 2024.

Nanda, N., Chan, L., Lieberum, T., Smith, J., and Steinhardt, J. Progress measures for grokking via mechanistic interpretability. *The Eleventh International Conference on Learning Representations*, 2023.

Power, A., Burda, Y., Edwards, H., Babuschkin, I., and Misra, V. Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*, 2022.

Purwins, H., Li, B., Virtanen, T., Schlüter, J., Chang, S.-Y., and Sainath, T. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219, 2019.

Rubey, M., Stump, C., et al. FindStat - The combinatorial statistics database. http://www.FindStat.org. URL http://www.FindStat.org. Accessed: July 4, 2025.

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

Stander, D., Yu, Q., Fan, H., and Biderman, S. Grokking group multiplication with cosets. *arXiv preprint arXiv:2312.06581*, 2023.

Stanley, R. P. *Enumerative combinatorics, volume 1*. Cambridge studies in advanced mathematics, 2011.

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. Accessed: [insert your access date].

Wu, W., Jaburi, L., Drori, J., and Gross, J. Unifying and verifying mechanistic interpretations: A case study with group operations. *arXiv preprint arXiv:2410.07476*, 2024.

Yip, C. H., Agrawal, R., Chan, L., and Gross, J. Modular addition without black-boxes: Compressing explanations of mlps that compute numerical integration. *arXiv preprint arXiv:2412.03773*, 2024.

Zhang, Y., Backurs, A., Bubeck, S., Eldan, R., Gunasekar, S., and Wagner, T. Unveiling transformers with lego: a synthetic reasoning task. *arXiv preprint arXiv:2206.04301*, 2022.

Zhong, Z., Liu, Z., Tegmark, M., and Andreas, J. The clock and the pizza: Two stories in mechanistic explanation of neural networks. *Advances in Neural Information Processing Systems*, 36, 2024.

6. Appendix

6.1. Other permutation concepts

Stack-sorting A permutation is *stack sortable* if it can be sorted by the following algorithm:

(1) Initialize an empty stack and empty final output

(2) For each element x in the input permutation: (a) while the stack is nonempty and x is larger than the last item added to the stack, pop this item from the stack to the output (b) put x onto the stack

(3) If the stack is nonempty, add remaining stack to the output.

When we apply this algorithm to the permutation 25341, we get the permutation 23145. After two more stack sorts we get the identity permutation, so the number of stack sorts needed to sort the permutation is 3.

6.2. Model architecture details

MLP models Small MLPs were trained on different combinations of input representation and statistic for a classification task. The MLP architecture has 3 hidden layers with 128, 128, and 64 neurons respectively, each employing ReLU activation and dropout for regularization. A hyperparameter sweep was performed over learning rate ($1e-5, 1e-4, 3e-4,$ or $1e-3$), dropout ($0.1, 0.3,$ or 0.5), and weight-decay values ($0.0001, 0.001,$ or 0.01) and then five training iterations run using the best hyperparameters for each. Before training, all of the representations were transformed using a one-hot encoding of each element, and then the matrix flattened into a single vector for consistency (with the exception of the permutation matrix which is already the one-hot-encoded one line representation, and just needs to be flattened).

Transformers We trained encoder-only transformers, performing a hyperparameter sweep over the encoder dimension (40 or 80), depth ($4, 6,$ or 8), and number of attention heads (4 or 8). The transformer models were also trained five times with the best hyperparameters, for each combination of input permutation representation and classification task.

6.3. Additional model performance results

Figures 3 and 4 show the accuracy of MLP models across multiple permutation sizes (values of N), separated by classification tasks. Similarly for the transformer models, Figures 5 and 6 show the accuracy across values of N, for each classification task. Figures 7 and 8 give MLP and transformer classification results organized by representation-task pairs for $N = 6$. Figures 9 and 10 give MLP and transformer classification results organized by representation-task pairs for $N = 7$.

6.4. Class distributions of statistics

Tables 2, 3, 4, and 5 give distributions of the various statistics by class for $N = 7$. The statistics are separated according to commonality in their number of classes.

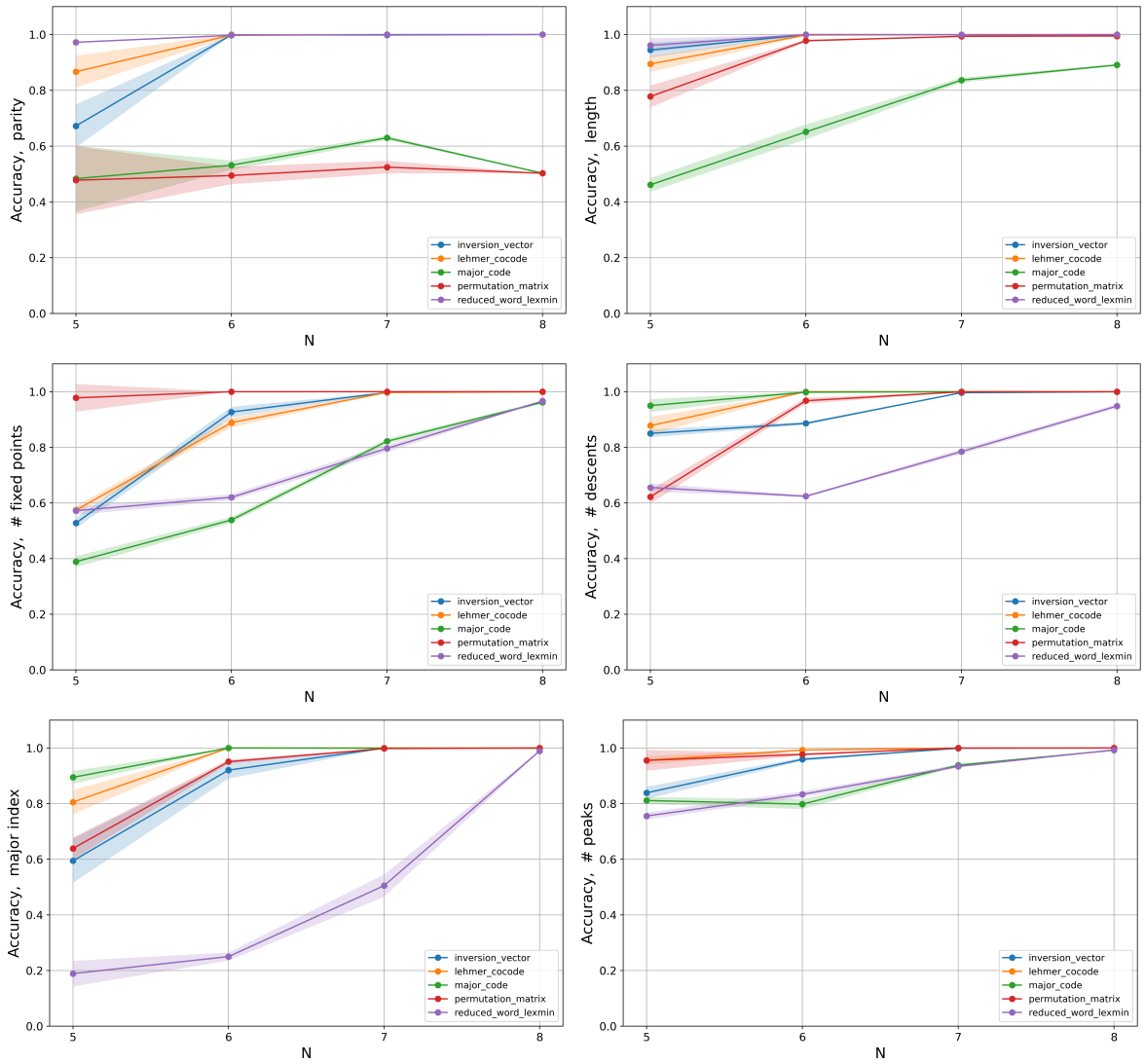

Figure 3. Accuracy of **MLP** classifiers on the parity, length, counting fixed points, counting descents, major index, and number of peaks tasks. Color bands indicate the standard deviation of the mean accuracy.

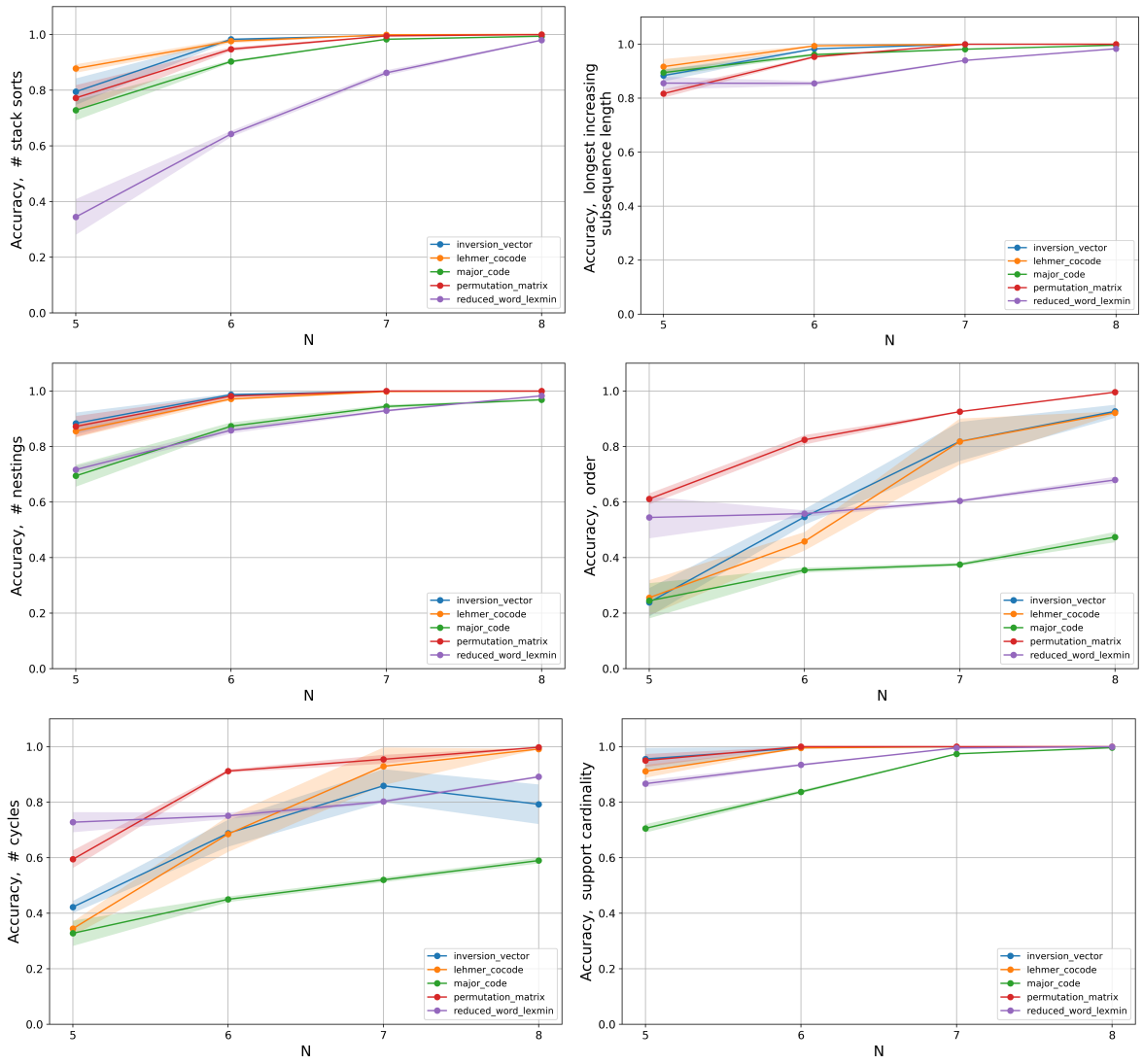

Figure 4. Accuracy of **MLP** classifiers on the number of stack sorts, length of longest increasing subsequence, number of nestings, order, number of cycles, and support cardinality tasks. Color bands indicate the standard deviation of the mean accuracy.

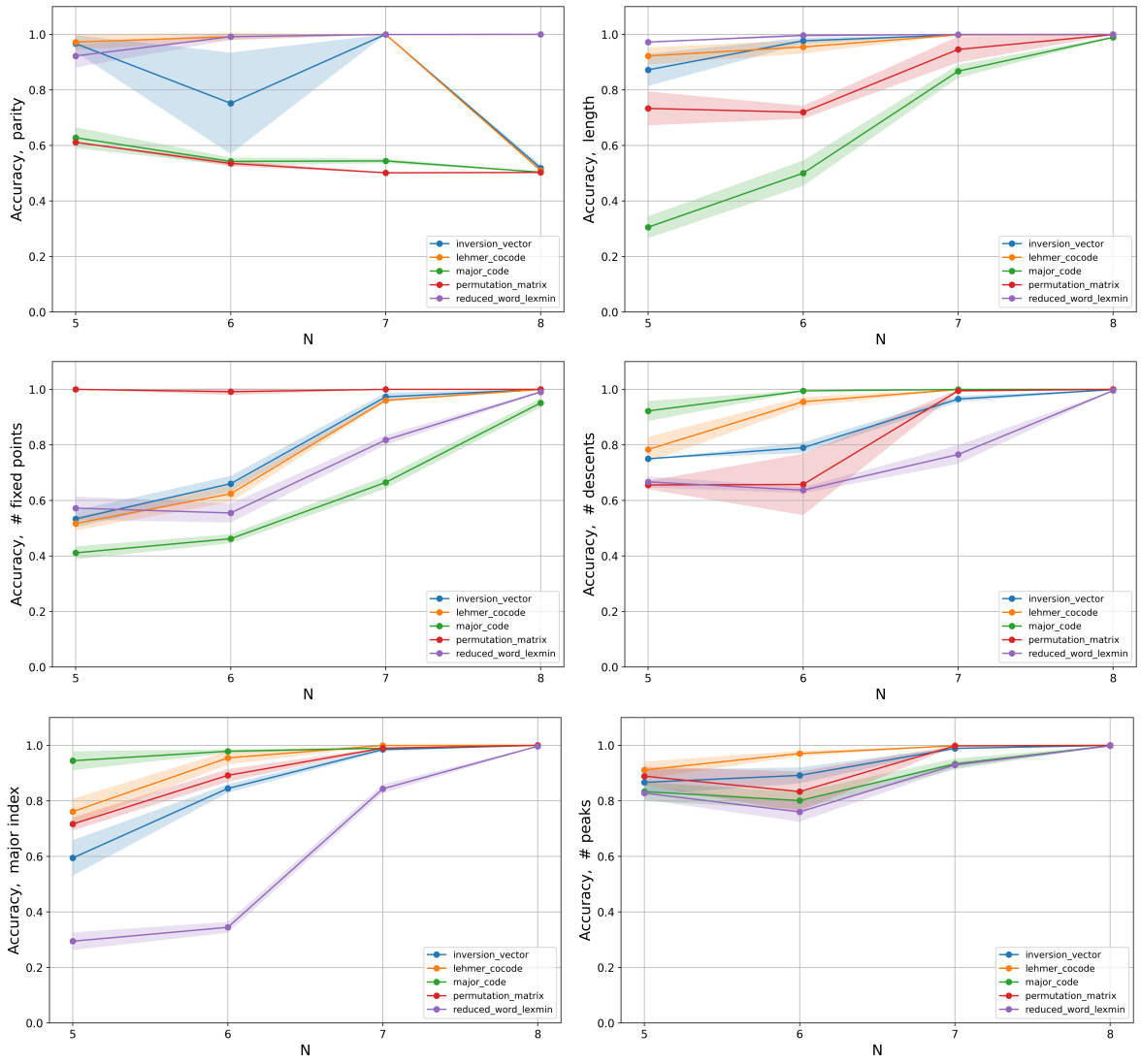

Figure 5. Accuracy of **transformer** classifiers on the parity, length, counting fixed points, counting descents, major index, and number of peaks tasks. Color bands indicate the standard deviation of the mean accuracy.

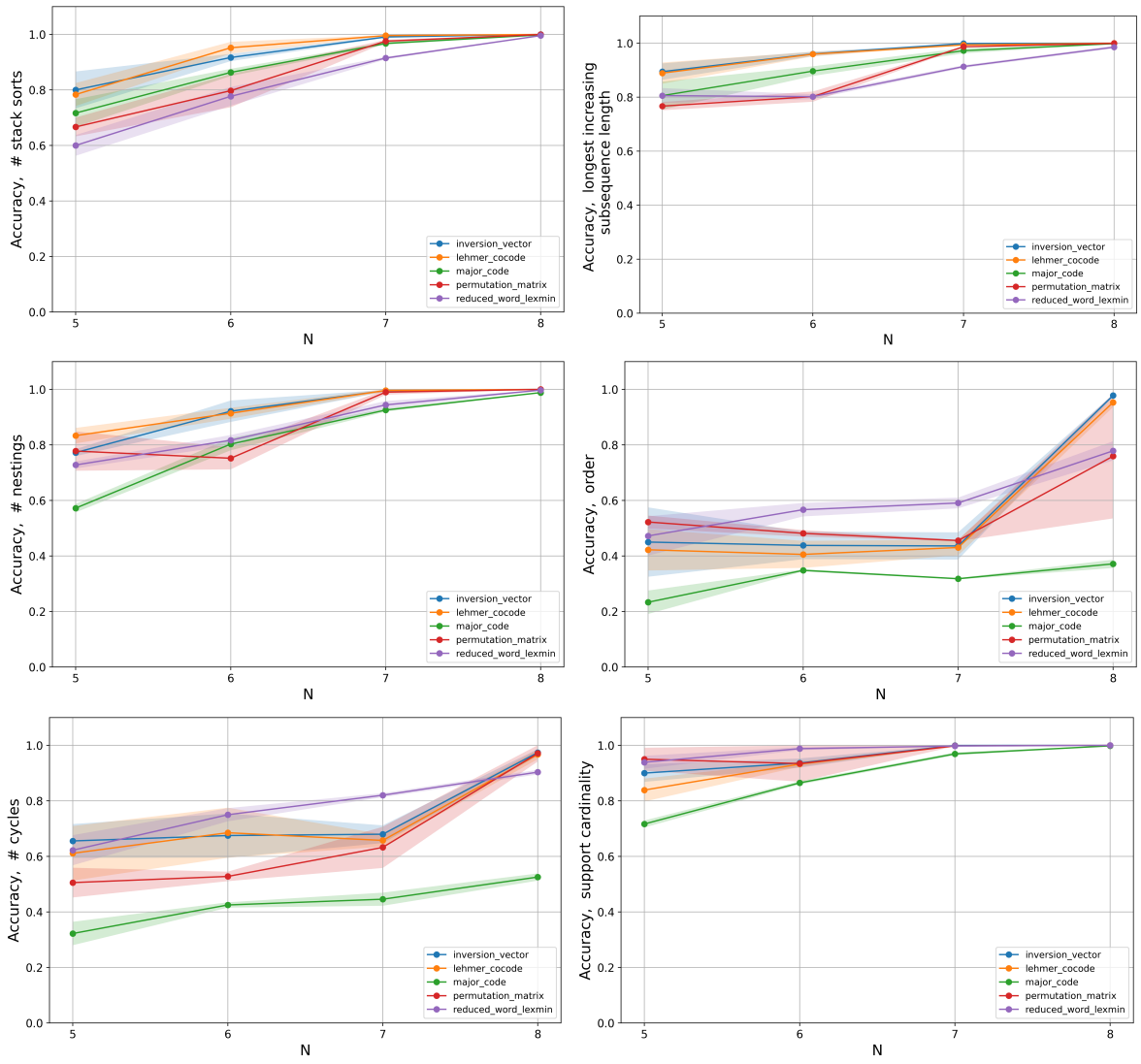

Figure 6. Accuracy of **transformer** classifiers on the number of stack sorts, length of longest increasing subsequence, number of nestings, order, number of cycles, and support cardinality tasks. Color bands indicate the standard deviation of the mean accuracy.

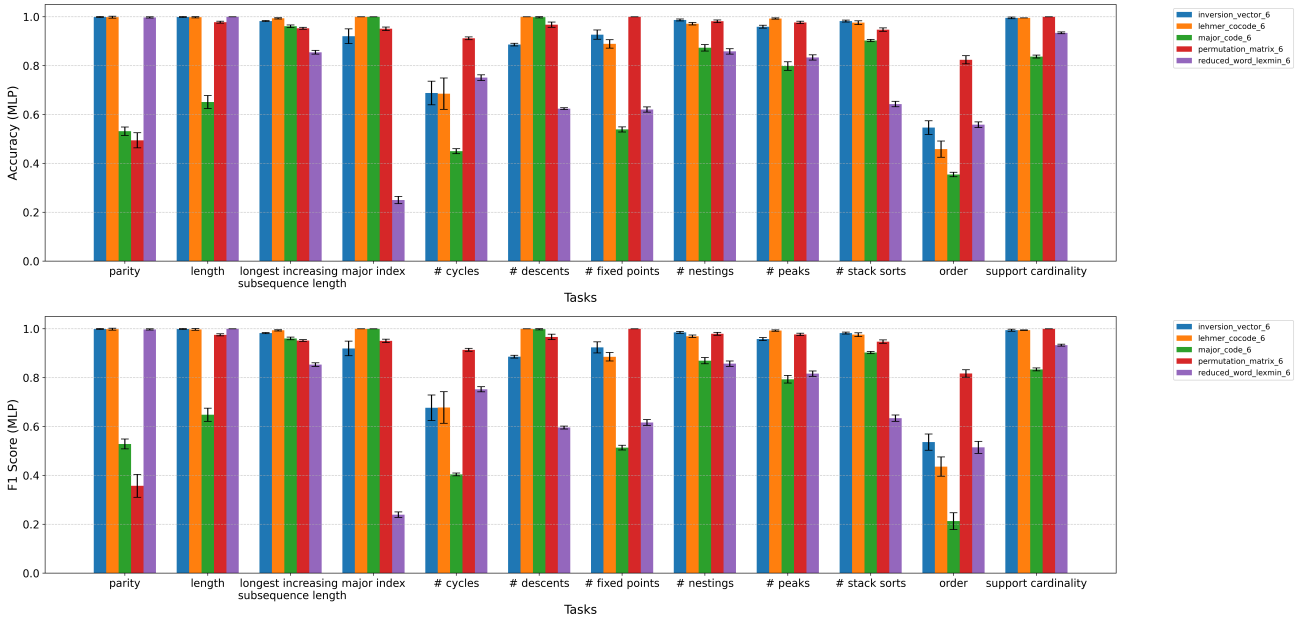

Figure 7. Average accuracy (top) and F1 scores (bottom) for the **MLP models** of each representation, statistic pair. Results are averaged over five training runs of each classifier and the standard deviation is indicated by error bars. Permutations from S_6.

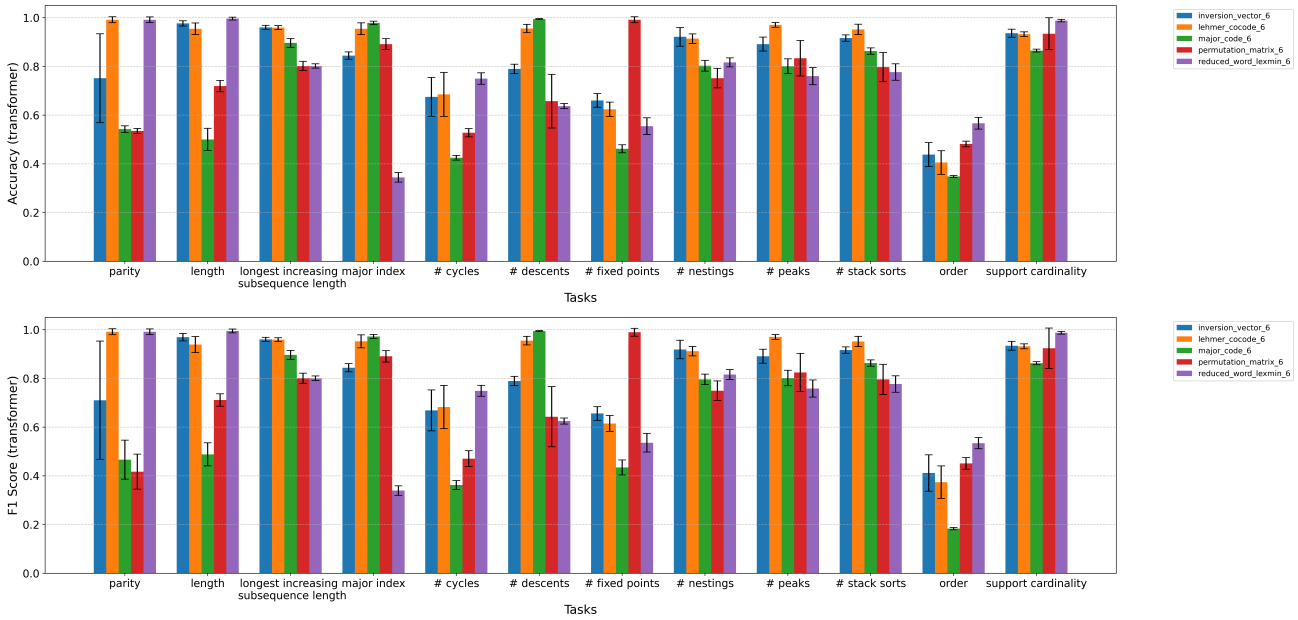

Figure 8. Average accuracy (top) and F1 scores (bottom) for the **transformer models** of each representation, statistic pair. Results are averaged over five training runs of each classifier and the standard deviation is indicated by error bars. Permutations from S_6.

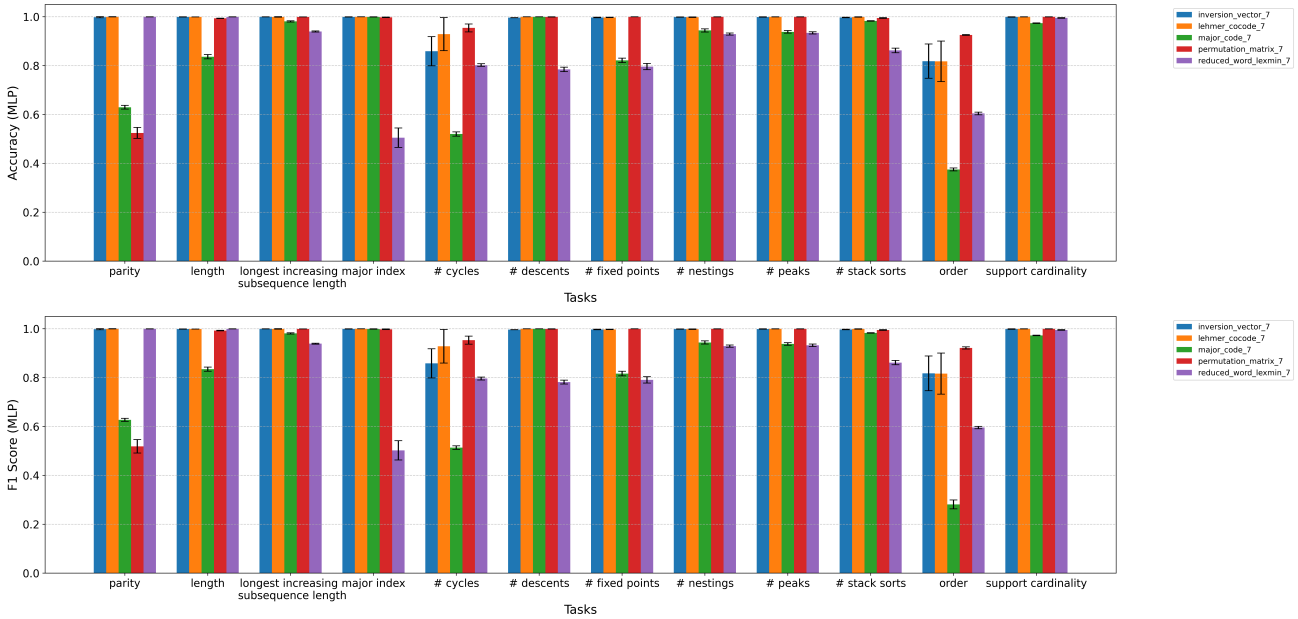

Figure 9. Average accuracy (top) and F1 scores (bottom) for the **MLP models** of each representation, statistic pair. Results are averaged over five training runs of each classifier and the standard deviation is indicated by error bars. Permutations from S_7.

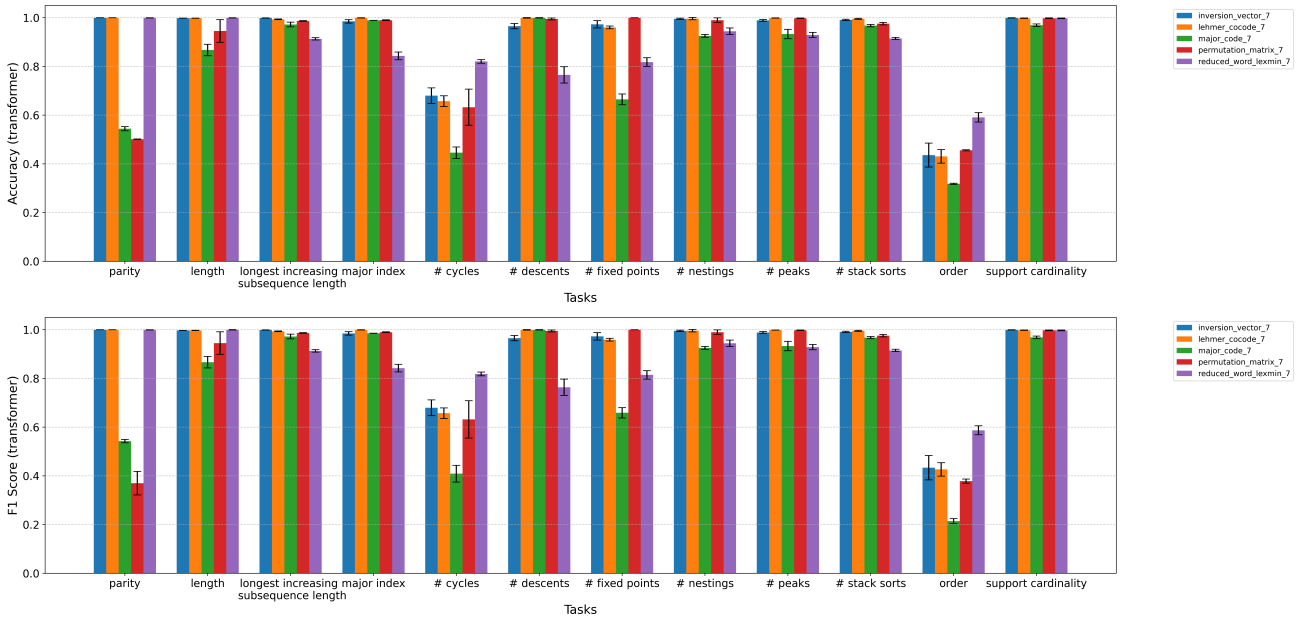

Figure 10. Average accuracy (top) and F1 scores (bottom) for the **transformer models**. Results are averaged over five training runs of each classifier and the standard deviation is indicated by error bars. Permutations from S_7.

task		# classes	0	1	2	3	4	5	6	7
# fixed points	train	7	1306	1291	653	214	50	13	0	1
	test	7	548	564	271	101	20	8	0	0
# descents	train	7	1	85	840	1701	815	85	1	0
	test	7	0	35	351	715	376	35	0	0
# stack sorts	train	7	1	313	1051	1094	665	313	91	0
	test	7	0	115	458	462	317	131	29	0
longest inc subseq length	train	7	0	1	294	1595	1317	295	25	1
	test	7	0	0	134	737	504	126	11	0
# cycles	train	7	0	507	1243	1142	508	114	13	1
	test	7	0	213	521	482	227	61	8	0
support cardinality	train	7	1	4	17	57	237	820	2392	0
	test	7	0	2	8	35	90	322	1055	0

Table 2. Distribution of statistics (number of fixed points, number of descents, number of stack sorts, longest increasing subsequence length, number of cycles, and support cardinality) by class for $N = 7$

Task		# classes	0	1	2	3	4	5	6	7	8	9	10	12
# nestings	train	10	303	733	880	765	465	247	97	29	8	1	–	–
	test	10	126	268	394	327	235	105	43	13	1	0	–	–
order	train	9	–	1	150	249	581	368	1017	507	0	0	367	288
	test	9	–	0	81	101	259	136	453	213	0	0	137	132

Table 3. Distribution of statistics (number of nestings and order) for $N = 7$

task		# classes	0	1	2	3
parity	train	2	1764	1764	–	–
	test	2	756	756	–	–
number peaks	train	4	45	1281	2012	190
	test	4	19	543	868	82

Table 4. Distribution of statistics (parity and number of peaks) for $N = 7$

task		classes	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
length	train	22	1	4	15	29	64	114	192	270	336	380	412	379	367	323	234	172	110	75	28	17	5	1
	test	22	0	2	5	20	34	55	67	89	119	151	161	194	164	132	125	87	59	23	21	3	1	0
major	train	22	1	4	15	37	76	109	190	252	310	376	396	403	374	318	240	185	118	77	29	11	6	1
index	test	22	0	2	5	12	22	60	69	107	145	155	177	170	157	137	119	74	51	21	20	9	0	0

Table 5. Distribution of length and major index for $N = 7$