Randomized Positional Encodings Boost Length Generalization of Transformers

Anian Ruoss[*1] **Grégoire Delétang**[*1] **Tim Genewein**[1] **Jordi Grau-Moya**[1]

Róbert Csordás[†2] **Mehdi Bennani**[1] **Shane Legg**[1] **Joel Veness**[1]

Abstract

Transformers have impressive generalization capabilities on tasks with a fixed context length. However, they fail to generalize to sequences of arbitrary length, even for seemingly simple tasks such as duplicating a string. Moreover, simply training on longer sequences is inefficient due to the quadratic computation complexity of the global attention mechanism. In this work, we demonstrate that this failure mode is linked to positional encodings being out-of-distribution for longer sequences (even for relative encodings) and introduce a novel family of positional encodings that can overcome this problem. Concretely, our randomized positional encoding scheme simulates the positions of longer sequences and randomly selects an ordered subset to fit the sequence's length. Our large-scale empirical evaluation of 6000 models across 15 algorithmic reasoning tasks shows that our method allows Transformers to generalize to sequences of unseen length (increasing test accuracy by 12.0% on average).

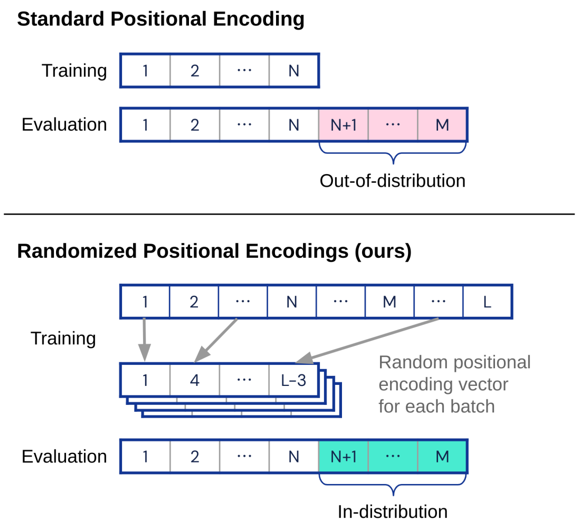

Standard Positional Encoding
Training
1
2
···
N
Evaluation
N+1
M
Out-of-distribution
Randomized Positional Encodings (ours)
L
4
L−3
Random positional encoding vector for each batch
In-distribution

Figure 1: **Test-time evaluation with longer inputs.** The standard positional encoding vector has values larger than those observed during training. Our approach avoids this problem by assigning a random (ordered) positional encoding vector using the full range of possible test positions to each training example.

1 Introduction

Transformers are emerging as the new workhorse of machine learning as they underpin many recent breakthroughs, including sequence-to-sequence modeling (Vaswani et al., 2017), image recognition (Dosovitskiy et al., 2021), and multi-task learning (Reed et al., 2022). However, recent work (Delétang et al., 2023) demonstrated that Transformers fail to generalize to longer sequences on seemingly simple tasks such as binary addition. Thus, while certain problems can be solved without length generalization, algorithmic reasoning generally requires this ability, similar to many real-world settings such as online or continual learning.

While the Transformer's attention mechanism can recognize complex relationships amongst tokens in the input sequence, it is limited by its lack of positional awareness. Thus, the input sequence is generally augmented with *positional encodings* to inject position information into the computation. However, current approaches only consider positions up to the maximum training sequence length N, and thus all the positions $N+1, \dots, M$ for test sequences of length up to M will appear out-of-distribution during evaluation (top of Fig. 1).

This work We introduce a novel family of *randomized positional encodings*, which significantly improves Transformers' length generalization capabilities on algorithmic reasoning tasks. Our approach is compatible with any existing positional encoding scheme and augments the existing methods by subsampling an ordered set of positions from a much larger range of positions than those observed during training or evaluation (i.e., up

[*]Equal contribution. [1]DeepMind. [2]The Swiss AI Lab, IDSIA, USI & SUPSI. [†]Work performed while the author was at DeepMind. Correspondence to {anianr, gdelt}@deepmind.com.

to $L \gg M$; bottom of Fig. 1). Thus, over the course of training, the Transformer will learn to handle very large positional encodings and, therefore no longer encounter out-of-distribution inputs during evaluation. Importantly, our method leaves in-domain generalization performance unaffected and is also significantly more efficient than the naive approach of simply training the Transformer on longer sequences. Our main contributions are:

- A novel family of positional encoding schemes that significantly improves the length generalization capabilities of Transformers, while leaving their in-domain generalization performance unaffected.
- A large-scale empirical evaluation on a wide range of algorithmic reasoning tasks showing the superiority of our method over prior work (an increase of the test accuracy by 12.0% on average and up to 43.5% on certain tasks).
- An open-source implementation of our method, available at `https://github.com/deepmind/randomized_positional_encodings`.

2 Related Work

Our work is most closely related to the growing line of research on Transformers' positional encodings. The first approaches simply added a transformation of the tokens' positions, e.g., scaled sinusoids (Vaswani et al., 2017) or learned embeddings (Gehring et al., 2017), to the embeddings of the input sequence. Dai et al. (2019) subsequently showed that computing the attention (at every layer) using the relative distances between the key and query vectors improves the modeling of long-term (inter-context) dependencies. Similarly, Su et al. (2021) proposed to inject position information by rotating the key-query products according to their relative distances. Finally, Press et al. (2022) improved the length generalization on natural language processing tasks by adding a constant bias to each key-query attention score (proportional to their distance). However, as our experiments in Section 4 will show, these approaches fail at length generalization on algorithmic reasoning tasks, which is precisely the goal of our work.

A concurrent work developed randomized learned positional encodings (Li and McClelland, 2022), which are a special case of our family of randomized positional encodings. We also note that the necessity of feature and position randomization for length generalization has been discussed in the context of graph neural networks, which subsume Transformers (Ibarz et al., 2022; Sato et al., 2021). Finally, Liu et al. (2020b) proposed to model the position information as a continuous dynamical system in an effort to handle sequences longer than those seen during training time.

Our work is also related to the research area on improving the systematic (length) generalization capabilities of Transformers (Ontañón et al., 2022), which includes approaches investigating embedding scaling or early stopping (Csordás et al., 2021), adaptive computation time (Dehghani et al., 2019), geometric attention with directional positional encodings and gating (Csordás et al., 2022), and hierarchical reinforcement learning (Liu et al., 2020a). Such length generalization studies are often conducted in the context of formal language theory, and we evaluate our method on the recent benchmark by Delétang et al. (2023), which unifies a large body of work on Transformers' capability to recognize formal languages (Ackerman and Cybenko, 2020; Bhattamishra et al., 2020; Ebrahimi et al., 2020; Hahn, 2020; Hao et al., 2022; Merrill, 2019; Merrill and Sabharwal, 2022).

3 Randomized Positional Encodings

Unlike RNNs (Elman, 1990), which are unrolled over tokens one step at a time, Transformers process large chunks of the input sequence in parallel via global attention (Vaswani et al., 2017). As a result, Transformers do not need to "remember" previous tokens, but they do have to break the permutation-invariance of the attention mechanism. To that end, the embeddings of the input sequence are generally augmented with positional encodings. For example, the vanilla Transformer adds the following positional encodings to the embedded input sequence before passing it to the attention layers:

$$\mathrm{PE}(\mathrm{pos}, 2i) = \sin\left(\frac{\mathrm{pos}}{10000^{\frac{2i}{d_{\mathrm{model}}}}}\right), \quad (1)$$

$$\mathrm{PE}(\mathrm{pos}, 2i+1) = \cos\left(\frac{\mathrm{pos}}{10000^{\frac{2i}{d_{\mathrm{model}}}}}\right), \quad (2)$$

where pos is the token's position in the sequence, $d_{\mathrm{model}} \in \mathbb{N}$ is the dimension of the input embedding, and $i \in \{1, 2, \ldots, d_{\mathrm{model}}/2\}$.

While positional encodings generally succeed at inducing the required positional information for sequences of fixed length, they are one of the main failure modes preventing length generalization. Concretely, for a Transformer with standard positional encodings trained on a curriculum of sequences of maximum length N, test sequences of length $M > N$ will shift the distribution of the resultant positional encodings away from those seen in training, with the shift getting increasingly large as M grows. To address this, we propose a randomized encoding scheme, which relies only on order information, and can be expected to generalize up to sequences of length M, where $N < M \leq L$, with a configurable hyperparameter L.

Randomized positional encodings We assume that each training step will perform a step of loss minimization on a batch of data of fixed size. Let $\mathcal{U}(S)$ denote the discrete uniform distribution over set S, and let $P_k := \{S \subseteq \{1, \dots, L\} \mid |S| = k\}$. For each training step, we first sample a random length $n \sim \mathcal{U}(\{1, \dots, N\})$ (following Delétang et al., 2023) and then a random set of indices $I \sim \mathcal{U}(P_n)$. We then sort I in ascending order, such that $I = \{i_1, \dots, i_n\}$ for $i_1 < i_2 < \dots < i_n$, noting that I is sampled without replacement. Finally, we compute our randomized positional encoding for token $1 \leq j \leq N$ as $\mathrm{RPE}(j, \cdot) := \mathrm{PE}(i_j, \cdot)$. At test time, when processing a sequence of length $M > N$, we use the same procedure but for all token positions $1 \leq j \leq M$. The intuition behind our method is to preserve the known good properties of relative encoding but in a way that is independent of the maximum training length N and thus allows generalization to longer sequences at test time.

When applying our randomized positional encoding scheme, we subsample the extended positions only once per batch and not individually for every sequence. For the $\sin$ / $\cos$ (Vaswani et al., 2017), learned (Gehring et al., 2017), and RoPE encodings (Su et al., 2021), we apply our method as described above, i.e., we directly replace the original token positions with their sampled counterpart. For the relative encoding (Dai et al., 2019), we compute the relative distances between the sampled positions instead of the original positions. Finally, for ALiBi (Press et al., 2022), we sample the bias values from the set of extended positions.

As a consequence, our tokens' positional encodings are no longer directly related to their exact position (the encodings even change during training as they are resampled at every step). However, since we maintain the order of the encodings, the Transformer can still learn to extract the relevant positional information from the subsampled encodings. Indeed, we validate the necessity of ordering the sampled positions in our ablation study in Appendix B.1. Thus, the success of our encoding scheme offers an interesting insight into the inductive biases of the Transformer architecture.

As we will show in Section 4, our randomized encodings trained only on lengths up to N perform the same on sequences of length M as prior approaches trained on lengths up to M. Therefore, our method demonstrates that Transformers can be efficiently trained on short sequences as long as (i) the longer sequences share the same structure and (ii) the longer positions are observed during training. Moreover, as the running time of global attention is $\mathcal{O}(\ell^2)$ for sequence length ℓ, our encoding scheme is significantly faster than directly training a model on long sequences. Furthermore, we also note that our randomized positional encoding scheme significantly boosts length generalization while leaving the in-domain generalization performance largely unaffected (see Fig. 4).

The main limitation of our approach is that the maximum test sequence length M has to be known in advance to choose $L \gg M$. However, our method is compatible with a wide range of values for L (see Appendix B.1), and we note that this is a much weaker assumption than that required for the naive approach of simply training on longer sequences. However, note that if L is chosen to be much larger than N or M, it is theoretically unlikely for the model to encounter enough unique indices during training, likely leading to poor performance (both in- and out-of-distribution).

4 Experimental Evaluation

Problem setup We closely follow the experiment setup of Delétang et al. (2023) and evaluate our method on a wide range of algorithmic reasoning tasks such as modular arithmetic, reversing/duplicating a string, binary addition/multiplication, and bucket sort. The tasks are derived from formal language recognition and thus grouped according to the Chomsky hierarchy (Chomsky, 1956), which partitions languages into regular (R), context-free, context-sensitive (CS), and recursively enumerable. Regular tasks can be solved by a finite-state automaton (FSA), deterministic context-free (DCF) tasks can be solved by an FSA with access to a deterministic stack, and

Table 1: Accuracy (in percentage) averaged over all test lengths and maximized over 10 random seeds and 3 learning rates. The random accuracy is 50%, except for Modular Arithmetic (Simple), Cycle Navigation, Bucket Sort, and Modular Arithmetic, where it is 20%. Our randomized method increases the test accuracy by 12.0% on average. The randomized learned encodings (denoted with $\star$) are equivalent to label-based encodings (Li and McClelland, 2022). † denotes permutation-invariant tasks, which can be solved without positional information.

								Randomized (Ours)				
Level	**Task**	**None**	sin / cos	**Relative**	**ALiBi**	**RoPE**	**Learned**	sin / cos	**Relative**	**ALiBi**	**RoPE**	**Learned***
R	Even Pairs	50.4	50.9	96.4	67.3	51.0	50.7	**100.0**	**100.0**	81.5	**100.0**	97.5
	Modular Arithmetic (Simple)	20.1	20.5	21.8	24.2	21.6	20.2	25.7	**28.1**	21.2	25.5	21.1
	Parity Check†	51.9	50.5	51.8	51.7	51.3	50.3	**52.6**	52.2	50.3	52.3	**52.6**
	Cycle Navigation†	61.9	26.3	23.0	37.6	23.6	24.2	59.0	58.8	29.8	**73.6**	49.7
DCF	Stack Manipulation	50.3	50.1	53.6	57.5	51.2	49.2	72.8	**77.9**	70.6	68.2	69.1
	Reverse String	52.8	50.6	58.3	62.3	51.9	50.7	75.6	**95.1**	77.1	69.9	52.9
	Modular Arithmetic	31.0	28.3	30.3	32.5	25.1	25.1	33.8	**34.9**	31.3	32.7	31.9
	Solve Equation	20.1	21.0	23.0	25.7	23.1	20.4	24.5	**28.1**	22.0	24.5	22.1
CS	Duplicate String	52.8	50.7	51.7	51.3	50.9	50.8	72.4	**75.1**	68.9	68.9	53.0
	Missing Duplicate	52.5	51.3	54.0	54.3	56.5	51.0	52.5	**100.0**	79.7	88.7	52.7
	Odds First	52.8	51.6	52.7	51.4	51.3	50.6	65.9	**69.3**	64.7	65.6	52.7
	Binary Addition	50.1	49.8	54.3	51.4	50.4	49.8	64.4	**64.5**	56.2	60.2	61.7
	Binary Multiplication	49.9	50.1	**52.2**	51.0	50.2	49.6	52.1	50.1	50.5	51.7	51.9
	Compute Sqrt	50.2	50.1	52.4	50.9	50.5	50.2	52.5	**53.3**	51.2	52.3	52.0
	Bucket Sort†	23.7	30.1	91.9	38.8	30.6	25.9	**100.0**	**100.0**	99.6	99.6	99.5

CS tasks can be solved by an FSA with access to a bounded tape. Note that the relation to the Chomsky hierarchy is largely irrelevant for our work and only included for completeness. We evaluate our method on Delétang et al. (2023)'s benchmark as it is currently out of reach for Transformers and clearly demonstrates their failure to generalize on algorithmic reasoning tasks. We refer interested readers to the original paper for more details.

We consider the encoder-only model of the original seq-to-seq Transformer (Vaswani et al., 2017), as used in popular pre-trained language models such as BERT (Devlin et al., 2019) or Gopher (Rae et al., 2021). Thus, for tasks that require a multi-token output sequence $\boldsymbol{y}$ (e.g., duplicating a string), we pad the input sequence with $|\boldsymbol{y}|$ empty tokens and compute the entire Transformer output from the padded sequence (i.e., we do not use autoregressive sampling). We train the model on sequences of length sampled uniformly from $\mathcal{U}(1, N)$, with $N = 40$, and evaluate it on sequences of length $\{N + 1, \ldots, M\}$, with $M = 500$. We set the maximum position $L = 2048$ (and visualize the impact of other values on the performance in Appendix B.1). We report the accuracy averaged over all unseen sequence lengths, i.e., $N + 1, \ldots, M$, for the best-performing model out of 10 different parameter initialization seeds and three learning rates 1×10^{-4}, 3×10^{-4}, 5×10^{-4}. We use the same hyperparameters as Delétang et al. (2023) and provide the full experiment setup in Appendix A. We make our code publicly available at https://github.com/deepmind/randomized_positional_encodings.

Comparison to prior work We compare our method to a wide range of positional encodings: none, sin / cos (Vaswani et al., 2017), relative (Dai et al., 2019), ALiBi (Press et al., 2022), RoPE (Su et al., 2021), learned (Gehring et al., 2017), and label-based (Li and McClelland, 2022). Note that the label encodings proposed by Li and McClelland (2022) are equivalent to randomized learned positional encodings and thus subsumed by our method. We instantiate our randomized positional encoding scheme with all the above encodings and show the average test accuracy in Table 1 (with performance curves over test lengths in Appendix B.2). We observe that our randomized versions significantly increase the test accuracy across most tasks (by 12.0% on average and up to 43.5%). In particular, the randomized relative encoding solves tasks that were previously out of reach for prior work (e.g., Reverse String or Missing Duplicate).

Efficiency comparison We now show that our method allows us to train a model on short sequences and obtain a test accuracy above 90%, roughly 35.4 times faster than the naive approach of training a model on longer sequences. To that end, we train the randomized relative encodings on sequences up to length 40 and the classical relative positional encoding (Dai et al., 2019) on sequences

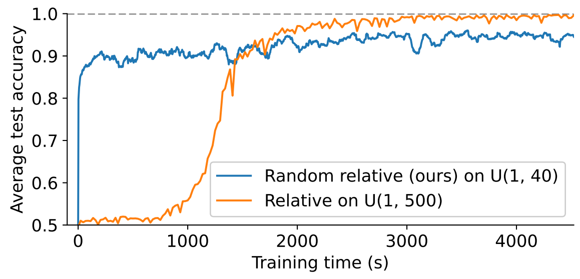


Figure 2: Average accuracy over unseen test lengths on the MISSING DUPLICATE task over training time (seconds) for two models: (i) our randomized relative positional encoding with a maximum training sequence length of 40, and (ii) the classical relative positional encoding but with a maximum training length of 500.

up to length 500 and show the test accuracy (averaged over lengths 41 to 500) in Fig. 2 over training time (in seconds). Our model obtains a strong test accuracy significantly faster due to the quadratic cost (in terms of sequence length) of global attention, which means that our model trains at 168.4 steps per second compared to 22.1 steps per second for the naive approach (on a NVIDIA V100 GPU).

5 Conclusion

We introduced a novel family of positional encodings that significantly improves the length generalization capabilities of Transformers. Our positional encodings are based on the insight that conventional positional encodings will be out-of-distribution when increasing the sequence length. Thus, to overcome this issue, we randomly sample our encodings from a wider range than the lengths seen at test time while keeping the order. Our large-scale empirical evaluation demonstrates that our method significantly outperforms prior work in terms of length generalization while offering superior computational performance over the naive approach of training the model on longer sequences.

Limitations

While our work shows promising results in improving the generalization capabilities of Transformers to sequences of arbitrary length, some limitations must be considered. First, our evaluation is confined to synthetic algorithmic reasoning tasks, which may not fully capture the complexity and diversity of natural language. We focused on synthetic datasets since they showed clear and somewhat surprising limitations of Transformer architectures (Delétang et al., 2023). However, the generalizability of our approach to other tasks and domains remains an open question, and additional research, such as evaluation on SCAN (Lake and Baroni, 2018), CFQ (Keysers et al., 2020), COGS (Kim and Linzen, 2020), or the Long Range Arena (Tay et al., 2021), is necessary to understand its potential in real-world applications. Second, our approach introduces a new hyperparameter – the maximum sequence position L. Although our experiments in Appendix B.1 show that our method's performance is largely unaffected by the precise value of L, practitioners may still have to tune the parameter depending on their specific problem domains. Third, we only isolate and ameliorate one failure mode of Transformer length generalization on synthetic datasets. However, there are other factors contributing to poor length generalization, such as attention becoming less peaked for longer sequences (Chiang and Cholak, 2022). Overall, we believe that our study's limitations offer several interesting directions for future research.

Acknowledgements

We thank Chris Cundy, Elliot Catt, Kevin Li, Laurent Orseau, Marcus Hutter, Petar Veličković, Vincent Dutordoir, and the anonymous reviewers for their helpful feedback.

References

Joshua Ackerman and George Cybenko. 2020. A survey of neural networks and formal languages. *arXiv:2006.01338*.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020. On the ability and limitations of transformers to recognize formal languages. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*.

David Chiang and Peter Cholak. 2022. Overcoming a theoretical limitation of self-attention. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*.

Noam Chomsky. 1956. Three models for the description of language. *IRE Trans. Inf. Theory*.

Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. 2021. The devil is in the detail: Simple tricks improve systematic generalization of transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. 2022. The neural data router: Adaptive control flow in transformers improves systematic generalization. In *The Tenth International Conference on Learning Representations*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc Viet Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Conference of the Association for Computational Linguistics*.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *7th International Conference on Learning Representations*.

Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. 2023. Neural networks and the chomsky hierarchy. In *The Eleventh International Conference on Learning Representations*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies,*.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations*.

Javid Ebrahimi, Dhruv Gelda, and Wei Zhang. 2020. How can self-attention networks recognize dyck-n languages? In *Findings of the Association for Computational Linguistics*.

Jeffrey L. Elman. 1990. Finding structure in time. *Cogn. Sci.*

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning*.

Michael Hahn. 2020. Theoretical limitations of self-attention in neural sequence models. *Trans. Assoc. Comput. Linguistics*.

Yiding Hao, Dana Angluin, and Robert Frank. 2022. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Trans. Assoc. Comput. Linguistics*.

Borja Ibarz, Vitaly Kurin, George Papamakarios, Kyriacos Nikiforou, Mehdi Bennani, Róbert Csordás, Andrew Joseph Dudzik, Matko Bosnjak, Alex Vitvitskyi, Yulia Rubanova, Andreea Deac, Beatrice Bevilacqua, Yaroslav Ganin, Charles Blundell, and Petar Velickovic. 2022. A generalist neural algorithmic learner. In *Learning on Graphs Conference, LoG 2022, 9-12 December 2022, Virtual Event*.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. 2020. Measuring compositional generalization: A comprehensive method on realistic data. In *8th International Conference on Learning Representations*.

Najoung Kim and Tal Linzen. 2020. COGS: A compositional generalization challenge based on semantic interpretation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*.

Brenden M. Lake and Marco Baroni. 2018. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *Proceedings of the 35th International Conference on Machine Learning*.

Yuxuan Li and James L. McClelland. 2022. Systematic generalization and emergent structures in transformers trained on structured tasks. *arXiv:2210.00400*.

Qian Liu, Shengnan An, Jian-Guang Lou, Bei Chen, Zeqi Lin, Yan Gao, Bin Zhou, Nanning Zheng, and Dongmei Zhang. 2020a. Compositional generalization by learning analytical expressions. In *Advances in Neural Information Processing Systems 33*.

Xuanqing Liu, Hsiang-Fu Yu, Inderjit S. Dhillon, and Cho-Jui Hsieh. 2020b. Learning to encode position for transformer with continuous dynamical model. In *Proceedings of the 37th International Conference on Machine Learning*.

William Merrill. 2019. Sequential neural networks as automata. *arXiv:1906.01615*.

William Merrill and Ashish Sabharwal. 2022. Log-precision transformers are constant-depth uniform threshold circuits. *arXiv:2207.00729*.

Santiago Ontañón, Joshua Ainslie, Zachary Fisher, and Vaclav Cvicek. 2022. Making transformers solve compositional tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Ofir Press, Noah Smith, and Mike Lewis. 2022. Train short, test long: Attention with linear biases enables input length extrapolation. In *The Tenth International Conference on Learning Representations*.

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, H. Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant M. Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew Johnson, Blake A. Hechtman, Laura Weidinger, Iason Gabriel, William S. Isaac, Edward Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. 2021. Scaling language models: Methods, analysis & insights from training gopher. *arXiv:2112.11446*.

Scott E. Reed, Konrad Zolna, Emilio Parisotto, Sergio Gómez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. 2022. A generalist agent. *Trans. Mach. Learn. Res.*

Ryoma Sato, Makoto Yamada, and Hisashi Kashima. 2021. Random features strengthen graph neural networks. In *Proceedings of the 2021 SIAM International Conference on Data Mining*.

Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. 2021. Roformer: Enhanced transformer with rotary position embedding. *arXiv:2104.09864*.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2021. Long range arena : A benchmark for efficient transformers. In *9th International Conference on Learning Representations*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*.

A Experimental Details

We use the experiment suite proposed by Delétang et al. (2023), which consists of 15 algorithmic reasoning tasks and is publicly available at `https://github.com/deepmind/neural_networks_chomsky_hierarchy` under the Apache 2.0 License. The tasks do not consist of fixed-size datasets but define training and testing distributions from which one can sample continuously. We train the models for 2 000 000 steps with a batch size of 128, which corresponds to 256 000 000 (potentially non-unique) training examples. At test time, we evaluate a single batch of size 500 for every sequence length in $\{41, \ldots, 500\}$, which corresponds to 230 000 testing examples. We use the Adam optimizer (Kingma and Ba, 2015) with gradient clipping and sweep over three learning rates: 1×10^{-4}, 3×10^{-4}, and 5×10^{-4}. Furthermore, for each task and positional encoding, we use 10 different parameter initialization random seeds.

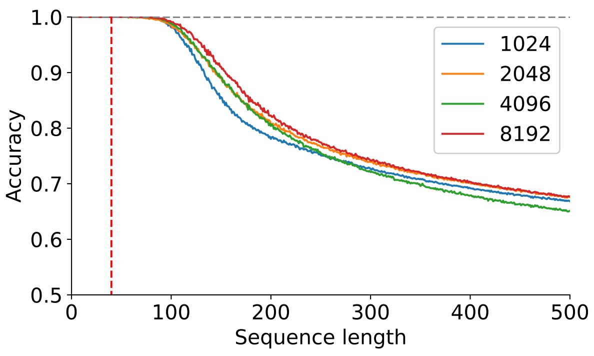
We consider the encoder-only Transformer architecture (Vaswani et al., 2017), with 5 blocks of 8 heads each and $d_{\text{model}} = 64$, which corresponds to 249 026 parameters (270 146 in the case of relative and randomized relative positional encodings). We run every task-encoding-hyperparameter triplet on a single NVIDIA V100 GPU from our internal cluster. As a result, we used $15 \text{ (tasks)} \cdot 13 \text{ (positional encodings)} \cdot 3 \text{ (learning rates)} \cdot 10 \text{ (seeds)} = 5850$ GPU-units for the results in Tables 1, 4 and 5 and Fig. 4. For the results in Fig. 2, we used an additional $2 \text{ (positional encodings)} \cdot 3 \text{ (learning rates)} \cdot 10 \text{ (seeds)} = 60$ GPU-units. Finally, for Fig. 3, we used $4 \text{ (maximum positions)} \cdot 3 \text{ (learning rates)} \cdot 10 \text{ (seeds)} = 120$ GPU-units, yielding a grand total of 6030 GPU-units. We report all running times in Table 2 and observe that our method induces a negligible computational overhead.

B Additional Results

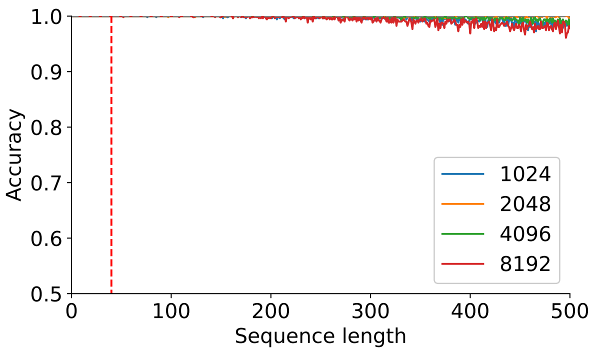
B.1 Ablation Study

In this section, we conduct an ablation study over the two main components of our method: (i) the maximum sampling position L, and (ii) the sorting of the subsampled positions.

We train the randomized relative positional encoding for a wide range of different maximum positions L: 1024, 2048, 4096, and 8192. Figure 3 shows that the test accuracy (averaged over all unseen sequence lengths) is largely unaffected by the value of L on the REVERSE STRING and MISSING DUPLICATE tasks. As a consequence, a practitioner wanting to apply our method will not have to carry out extensive tuning of this parameter (as long as it is larger than the maximum evaluation sequence length M, but not unreasonably large).



(a) REVERSE STRING (DCF)



(b) MISSING DUPLICATE (CS)

Figure 3: Sweep over the maximum position L for our randomized relative positional encodings. The test accuracy (averaged over unseen sequence lengths) is largely unaffected by the concrete value of L (for reasonably small values of L), showing the stability of our method. However, if L is much larger than the maximum training (N) or testing (M) sequence length, we expect the performance to degrade since it the model is unlikely to encounter enough unique indices during training time.

Next, we investigate the performance of our randomized $\sin / \cos$ positional encoding with and without sorting of the subsampled positions. Note that this experiment is meant as a "sanity-check" since we do not expect the Transformer to perform well without order information. Table 3 shows the test accuracy (averaged over all unseen sequence lengths) for the two versions of our method. We observe that sorting the positions is crucial, as it increases the test accuracy by 15.7% on average

Table 2: Mean and standard deviation of the running times (in hours) for all the positional encodings and tasks.

								Randomized (Ours)				
Level	**Task**	**None**	sin / cos	**Relative**	**ALiBi**	**RoPE**	**Learned**	sin / cos	**Relative**	**ALiBi**	**RoPE**	**Learned***
R	Parity Check†	0.86 ± 0.17	0.87 ± 0.17	1.63 ± 0.28	0.87 ± 0.17	1.41 ± 0.24	0.90 ± 0.18	0.92 ± 0.18	1.75 ± 0.29	0.94 ± 0.19	1.66 ± 0.31	1.12 ± 0.23
	Reverse String	1.17 ± 0.21	1.18 ± 0.22	2.61 ± 0.39	1.17 ± 0.22	2.01 ± 0.35	1.23 ± 0.23	1.24 ± 0.23	2.75 ± 0.41	1.27 ± 0.24	2.42 ± 0.43	1.62 ± 0.32
	Cycle Navigation†	0.86 ± 0.17	0.87 ± 0.17	1.62 ± 0.27	0.86 ± 0.17	1.41 ± 0.25	0.91 ± 0.18	0.92 ± 0.18	1.75 ± 0.29	0.94 ± 0.19	1.66 ± 0.31	1.12 ± 0.22
	Even Pairs	0.86 ± 0.17	0.87 ± 0.17	1.63 ± 0.27	0.86 ± 0.17	1.41 ± 0.24	0.91 ± 0.18	0.92 ± 0.18	1.75 ± 0.29	0.95 ± 0.19	1.65 ± 0.31	1.12 ± 0.22
DCF	Stack Manipulation	8.09 ± 0.97	8.00 ± 0.82	9.50 ± 0.89	8.07 ± 0.94	8.87 ± 0.84	8.46 ± 0.84	8.47 ± 0.88	10.04 ± 0.96	8.55 ± 0.90	10.61 ± 1.58	9.58 ± 1.12
	Modular Arithmetic	5.48 ± 0.63	5.55 ± 0.67	6.32 ± 0.81	5.50 ± 0.65	6.07 ± 0.69	5.69 ± 0.65	5.66 ± 0.64	6.56 ± 0.70	5.69 ± 0.65	6.41 ± 0.84	5.92 ± 0.80
	Binary Multiplication	1.83 ± 0.33	1.83 ± 0.30	2.86 ± 0.43	1.84 ± 0.31	2.32 ± 0.39	2.24 ± 0.35	2.23 ± 0.35	3.13 ± 0.43	2.24 ± 0.35	3.21 ± 0.51	2.88 ± 0.46
	Binary Addition	1.83 ± 0.32	1.82 ± 0.31	2.89 ± 0.42	1.81 ± 0.32	2.34 ± 0.39	2.22 ± 0.35	2.22 ± 0.35	3.17 ± 0.44	2.24 ± 0.35	3.29 ± 0.62	2.90 ± 0.49
CS	Binary Addition	1.83 ± 0.32	1.82 ± 0.31	2.89 ± 0.42	1.81 ± 0.32	2.34 ± 0.39	2.22 ± 0.35	2.22 ± 0.35	3.17 ± 0.44	2.24 ± 0.35	3.29 ± 0.62	2.90 ± 0.49
	Compute Sqrt	1.39 ± 0.24	1.40 ± 0.25	2.20 ± 0.34	1.40 ± 0.25	1.86 ± 0.30	1.73 ± 0.29	1.72 ± 0.29	2.43 ± 0.37	1.74 ± 0.30	2.53 ± 0.41	2.23 ± 0.38
	Solve Equation	5.60 ± 0.65	5.60 ± 0.67	6.41 ± 0.68	5.63 ± 0.66	6.14 ± 0.68	5.74 ± 0.65	5.78 ± 0.66	6.69 ± 0.76	5.83 ± 0.69	6.50 ± 0.80	6.01 ± 0.84
	Duplicate String	1.58 ± 0.28	1.59 ± 0.28	4.10 ± 0.54	1.58 ± 0.27	2.71 ± 0.40	1.64 ± 0.28	1.65 ± 0.29	4.24 ± 0.54	1.67 ± 0.29	3.18 ± 0.49	2.05 ± 0.38
	Modular Arithmetic (Simple)	0.99 ± 0.19	1.00 ± 0.19	1.74 ± 0.29	0.99 ± 0.19	1.51 ± 0.26	1.03 ± 0.20	1.05 ± 0.20	1.87 ± 0.31	1.06 ± 0.21	1.74 ± 0.31	1.23 ± 0.23
	Missing Duplicate	0.88 ± 0.17	0.90 ± 0.18	1.64 ± 0.27	0.88 ± 0.17	1.43 ± 0.26	0.93 ± 0.19	0.94 ± 0.19	1.78 ± 0.30	0.97 ± 0.19	1.66 ± 0.30	1.15 ± 0.23
	Odds First	1.17 ± 0.22	1.19 ± 0.22	2.61 ± 0.38	1.17 ± 0.22	2.00 ± 0.31	1.23 ± 0.23	1.24 ± 0.23	2.74 ± 0.40	1.26 ± 0.23	2.40 ± 0.39	1.59 ± 0.29
	Bucket Sort†	1.17 ± 0.23	1.18 ± 0.22	2.61 ± 0.43	1.16 ± 0.22	2.01 ± 0.34	1.22 ± 0.23	1.24 ± 0.23	2.74 ± 0.40	1.25 ± 0.23	2.40 ± 0.41	1.60 ± 0.30

Table 3: Accuracy (in percentage) averaged over all test lengths and maximized over 10 seeds and 3 learning rates for our randomized sin / cos positional encoding with and without sorting of the subsampled positions.

		Randomized sin / cos	
Level	**Task**	**w/o Sorting**	**w/ Sorting**
R	Even Pairs	50.4	**100.0**
	Modular Arithmetic (Simple)	20.0	**25.7**
	Parity Check†	52.2	**52.6**
	Cycle Navigation†	**59.3**	59.0
DCF	Stack Manipulation	50.4	**72.8**
	Reverse String	52.8	**75.6**
	Modular Arithmetic	31.0	**33.8**
	Solve Equation	20.2	**24.5**
CS	Duplicate String	52.8	**72.4**
	Missing Duplicate	**53.1**	52.5
	Odds First	52.8	**65.9**
	Binary Addition	50.0	**64.4**
	Binary Multiplication	49.9	**52.1**
	Compute Sqrt	50.2	**52.5**
	Bucket Sort†	23.7	**100.0**

and up to 76.3% on certain tasks. In fact, without sorting, our approach fails to beat the (baseline) random accuracy on all but the Cycle Navigation task, which is permutation-invariant (i.e., it can be solved without positional information). This confirms our intuition that the Transformer only needs to know the relative order of the positional encodings (and not their exact values), but that it fails to solve tasks when presented with positional encodings whose order does not correspond to the tokens' positions.

B.2 Comparison to Prior Work

In Section 4, we compared our method to a wide range of positional encodings: none, sin / cos (Vaswani et al., 2017), relative (Dai et al., 2019), ALiBi (Press et al., 2022), RoPE (Su et al., 2021), learned (Gehring et al., 2017), and label-based (Li and McClelland, 2022). Here, we provide additional results for these experiments, as well as a comparison to the geometric attention and directional encodings of Csordás et al. (2022).

We recall that Table 1 showed the test accuracy maximized over the 10 parameter initialization seeds and the three different learning rates. We reported the maximum following the experiment setup in Delétang et al. (2023), which investigates whether an architecture is capable of solving a task at all (and not on average). However, we also report the means and standard deviations (over the random seeds) in Table 4 for the best-performing learning rate. We observe that our randomized positional encoding also significantly outperform their original counterparts on average. We visualize the test accuracy per sequence length in Fig. 4.

We highlight the case of learned positional encodings, which fail to beat the random accuracy baseline (cf. Tables 1 and 4). This is because the columns of the embedding matrix corresponding to the positions that are larger than the maximum training length N are not learned during training and are thus entirely random. In contrast, our randomized version of the learned encodings considers all possible embedding columns during training and thus achieves non-trivial to strong length generalization on most tasks.

Finally, we also compare our method to a variant of the Neural Data Router (NDR) (Csordás et al., 2022), which was developed to improve the systematic generalization capabilities of Transformers. We only consider the most related aspects of the NDR architecture, i.e., the geometric attention and the directional encoding (we do not use gating or shared layers). Table 5 compares the test accuracy of geometric attention and directional encodings

Table 4: Means and standard deviations (computed over random seeds) of the score (accuracy averaged over all test lengths) for the results of the main experiment (see Table 1). The random accuracy is 50%, except for CYCLE NAVIGATION, BUCKET SORT, and the modular arithmetic tasks, where it is 20%. We denote permutation-invariant tasks, which can be solved without positional information, with †. Numbers in bold are the best performers, per task. These results underline the superiority of our method, and especially when applied to relative positional encodings.

								Randomized (Ours)				
Level	**Task**	**None**	sin / cos	**Relative**	**ALiBi**	**RoPE**	**Learned**	sin / cos	**Relative**	**ALiBi**	**RoPE**	**Learned***
R	EVEN PAIRS	50.1 ± 0.1	50.4 ± 0.2	67.6 ± 15.3	59.8 ± 3.2	50.4 ± 0.3	50.4 ± 0.2	99.7 ± 0.3	99.6 ± 0.6	71.4 ± 5.6	$\mathbf{100.0} \pm 0.0$	96.2 ± 0.7
	MODULAR ARITHMETIC (SIMPLE)	20.0 ± 0.0	20.2 ± 0.2	20.7 ± 0.5	23.2 ± 0.9	20.8 ± 0.5	20.1 ± 0.1	24.2 ± 1.4	$\mathbf{24.9} \pm 1.7$	20.8 ± 0.3	23.5 ± 1.6	20.2 ± 0.4
	PARITY CHECK†	50.4 ± 0.8	50.3 ± 0.2	50.4 ± 0.6	50.5 ± 0.6	50.4 ± 0.4	50.0 ± 0.1	51.1 ± 1.3	$\mathbf{51.4} \pm 0.5$	50.0 ± 0.2	50.4 ± 1.0	50.6 ± 0.9
	CYCLE NAVIGATION†	33.9 ± 10.5	23.8 ± 1.4	21.7 ± 0.8	31.1 ± 3.8	22.3 ± 0.9	21.0 ± 1.2	30.3 ± 10.7	45.9 ± 9.9	26.3 ± 2.4	$\mathbf{52.9} \pm 15.3$	31.9 ± 8.2
DCF	STACK MANIPULATION	50.2 ± 0.1	47.3 ± 1.9	50.1 ± 3.3	51.0 ± 8.0	49.6 ± 3.0	44.9 ± 3.7	69.2 ± 3.2	$\mathbf{71.7} \pm 4.7$	69.5 ± 1.1	66.0 ± 2.0	66.1 ± 2.5
	REVERSE STRING	52.7 ± 0.1	50.4 ± 0.1	54.2 ± 1.5	56.3 ± 2.6	51.2 ± 0.3	50.4 ± 0.2	72.9 ± 1.6	$\mathbf{77.1} \pm 6.6$	75.1 ± 1.3	67.7 ± 1.1	52.7 ± 0.2
	MODULAR ARITHMETIC	31.0 ± 0.1	24.3 ± 2.2	26.1 ± 2.0	28.1 ± 3.4	24.0 ± 2.4	22.3 ± 1.5	29.6 ± 4.6	28.8 ± 5.5	29.3 ± 1.6	28.6 ± 3.9	$\mathbf{30.3} \pm 2.6$
	SOLVE EQUATION	20.1 ± 0.0	20.9 ± 0.2	21.9 ± 0.7	23.6 ± 1.9	21.9 ± 0.6	20.2 ± 0.2	23.6 ± 0.5	$\mathbf{25.4} \pm 1.8$	21.1 ± 0.7	22.3 ± 1.6	21.1 ± 0.7
CS	DUPLICATE STRING	52.7 ± 0.1	50.4 ± 0.2	51.0 ± 0.4	51.0 ± 0.2	50.4 ± 0.2	50.4 ± 0.2	69.0 ± 2.9	$\mathbf{73.1} \pm 1.5$	67.9 ± 1.4	67.1 ± 2.0	52.8 ± 0.1
	MISSING DUPLICATE	51.4 ± 1.0	50.1 ± 0.6	51.1 ± 1.1	53.5 ± 0.4	53.9 ± 1.6	50.1 ± 0.4	50.4 ± 1.5	$\mathbf{91.4} \pm 9.8$	75.2 ± 3.4	73.2 ± 1.2	51.2 ± 1.4
	ODDS FIRST	52.7 ± 0.1	51.3 ± 0.2	51.5 ± 0.5	51.1 ± 0.2	50.8 ± 0.2	50.5 ± 0.1	62.5 ± 2.0	$\mathbf{65.9} \pm 1.6$	62.2 ± 1.4	62.9 ± 1.3	52.7 ± 0.1
	BINARY ADDITION	49.4 ± 0.3	47.3 ± 3.8	51.7 ± 1.3	48.5 ± 3.6	47.8 ± 5.4	48.9 ± 0.8	61.2 ± 1.7	$\mathbf{62.0} \pm 1.1$	54.3 ± 1.5	57.4 ± 1.2	59.9 ± 1.3
	BINARY MULTIPLICATION	49.8 ± 0.0	48.8 ± 1.0	50.2 ± 3.5	49.9 ± 2.3	49.6 ± 0.6	48.7 ± 1.7	$\mathbf{51.8} \pm 0.2$	39.1 ± 7.1	49.2 ± 1.2	45.7 ± 6.6	51.6 ± 0.2
	COMPUTE SQRT	50.2 ± 0.0	50.1 ± 0.0	51.5 ± 0.4	50.5 ± 0.2	50.3 ± 0.1	50.1 ± 0.1	51.9 ± 0.5	$\mathbf{52.4} \pm 0.6$	51.1 ± 0.1	51.8 ± 0.3	51.0 ± 0.8
	BUCKET SORT†	23.7 ± 0.0	25.6 ± 2.6	83.4 ± 6.6	29.3 ± 6.7	23.6 ± 3.8	20.7 ± 2.9	99.3 ± 0.4	$\mathbf{99.4} \pm 0.3$	98.8 ± 0.7	99.3 ± 0.3	98.9 ± 0.4

Table 5: Accuracy (in %) averaged over all test lengths for geometric attention with directional encoding.

		Max		Avg ± SD	
Level	**Task**	**Table 1**	**Geometric**	**Table 4**	**Geometric**
R	EVEN PAIRS	**100.0**	**100.0**	$\mathbf{100.0} \pm 0.0$	94.5 ± 8.8
	MODULAR ARITHMETIC (SIMPLE)	28.1	**43.6**	24.9 ± 1.7	$\mathbf{27.2} \pm 8.2$
	PARITY CHECK†	**52.6**	52.4	51.4 ± 0.5	$\mathbf{51.6} \pm 0.6$
	CYCLE NAVIGATION†	**73.6**	41.3	$\mathbf{52.9} \pm 15.3$	32.9 ± 4.7
DCF	STACK MANIPULATION	**77.9**	58.3	$\mathbf{71.7} \pm 4.7$	55.6 ± 2.3
	REVERSE STRING	**95.1**	65.2	$\mathbf{77.1} \pm 6.6$	59.3 ± 3.2
	MODULAR ARITHMETIC	34.9	**36.5**	30.3 ± 2.6	$\mathbf{32.8} \pm 2.8$
	SOLVE EQUATION	28.1	**31.7**	25.4 ± 1.8	$\mathbf{28.5} \pm 2.0$
CS	DUPLICATE STRING	**75.1**	58.6	$\mathbf{73.1} \pm 1.5$	54.9 ± 1.6
	MISSING DUPLICATE	**100.0**	64.4	$\mathbf{91.4} \pm 9.8$	60.3 ± 2.3
	ODDS FIRST	**69.3**	64.2	$\mathbf{65.9} \pm 1.6$	58.1 ± 2.6
	BINARY ADDITION	**64.5**	54.9	$\mathbf{62.0} \pm 1.1$	53.5 ± 1.5
	BINARY MULTIPLICATION	50.1	**53.6**	51.8 ± 0.2	$\mathbf{52.1} \pm 2.5$
	COMPUTE SQRT	53.3	**54.1**	$\mathbf{52.4} \pm 0.6$	52.3 ± 0.9
	BUCKET SORT†	**100.0**	78.3	$\mathbf{99.5} \pm 0.3$	57.7 ± 11.4

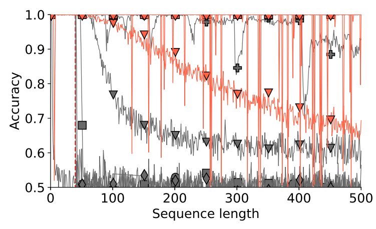
with the best results from Table 1 (for the maximum) and Table 4 (for the mean). We observe that our randomized positional encodings outperform the geometric attention overall (with a 9.7% higher maximum test accuracy on average) but not on all tasks. In particular, geometric attention performs substantially better on MODULAR ARITHMETIC (SIMPLE), which has an inherent locality bias, i.e., numbers closer to the operation symbols are generally more relevant, which can be captured by "radiating outwards" as geometric attention does.

B.3 Analysis

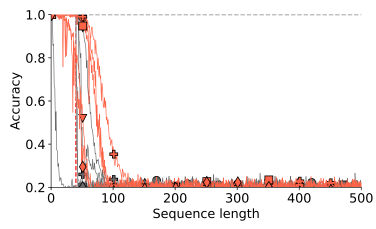
Analyzing the activations As illustrated in Fig. 1, the main intuition behind our randomized encodings is that they do not lead to out-of-distribution activations when evaluating on sequences longer than the maximal training length. We confirm this intuition in our analysis in Fig. 5, which shows a 2D projection of activations onto the first two principal components when evaluating on sequences of length 40 (i.e., the maximum training length N, shown in blue) and length 150 (i.e., the generalization regime, shown in orange), using the same transformation. While the activations of our randomized relative encoding strongly overlap for the training and the generalization regimes in all layers, the standard relative encoding leads to out-of-distribution activations for sequence length 150 in layers 3 and 4. We obtained qualitatively similar results for the $\sin / \cos$ and learned encodings.

To compute the results in Fig. 5, we generated 30 sequences of length 40 and 150 respectively, on the REVERSE STRING task and passed them through a well-trained model with either relative or randomized relative encodings. For each layer shown, we fitted a (non-whitened) 2D PCA on the activations obtained from sequence length 40 and projected all activations from sequence length 150 into two dimensions using the same transformations (yielding 30×40 and 30×150 activation-datapoints per layer). The random relative encoding (our method) attains an average accuracy of 1.0 and 0.994 on the 30 sequences of length 40 and 150, respectively. The standard relative encoding (the baseline) attains an average accuracy of 1.0 on sequence-length 40 and 0.596 on length 150, indicating the model's failure to generalize well under the standard relative encoding.

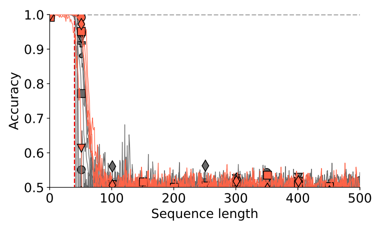
Analyzing the attention matrices We also analyze the attention matrices learned with the relative positional encoding and our corresponding random-



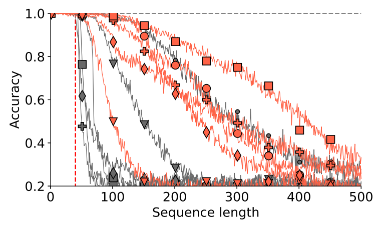
(a) EVEN PAIRS (R)



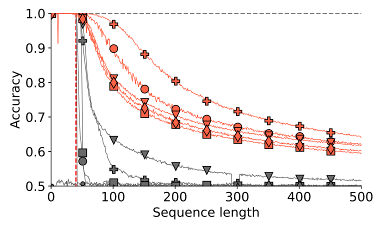
(b) MODULAR ARITHMETIC (SIMPLE) (R)



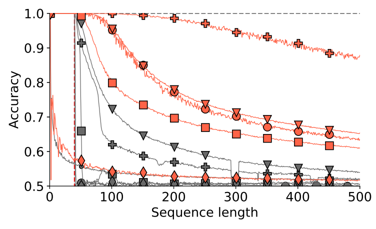
(c) PARITY CHECK (R)



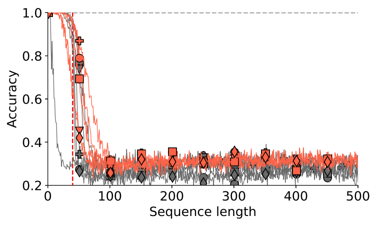
(d) CYCLE NAVIGATION (R)



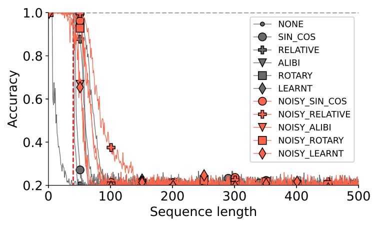
(e) STACK MANIPULATION (DCF)



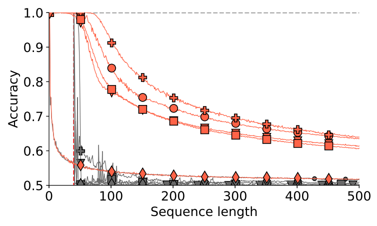
(f) REVERSE STRING (DCF)



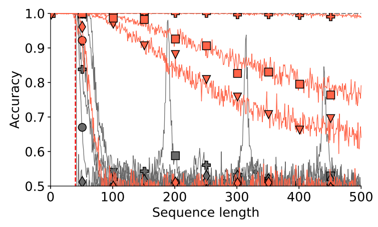
(g) MODULAR ARITHMETIC (DCF)



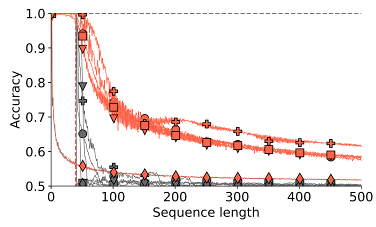
(h) SOLVE EQUATION (DCF)



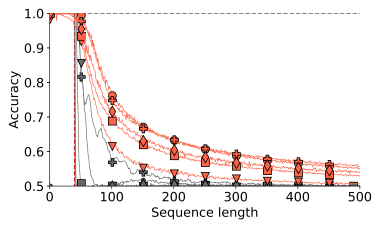
(i) DUPLICATE STRING (CS)



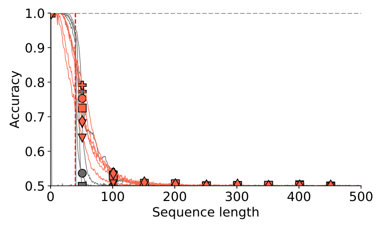
(j) MISSING DUPLICATE (CS)



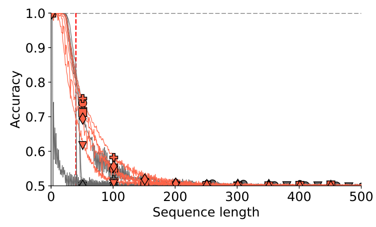
(k) ODDS FIRST (CS)



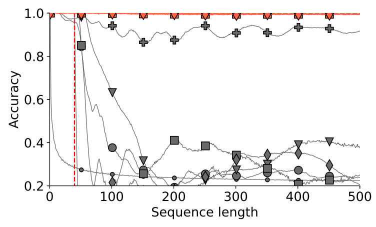
(l) BINARY ADDITION (CS)



(m) BINARY MULTIPLICATION (CS)

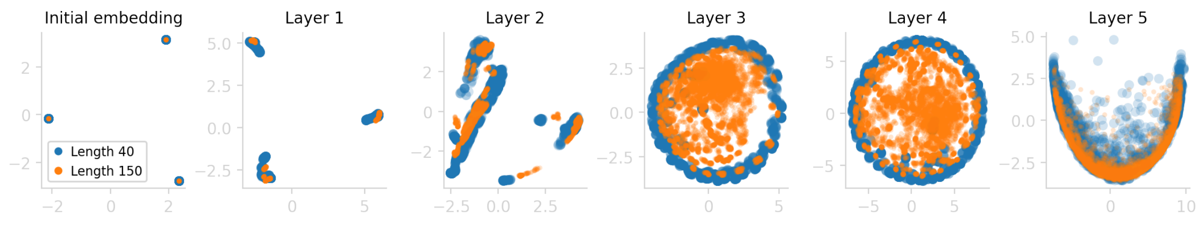


(n) COMPUTE SQRT (CS)

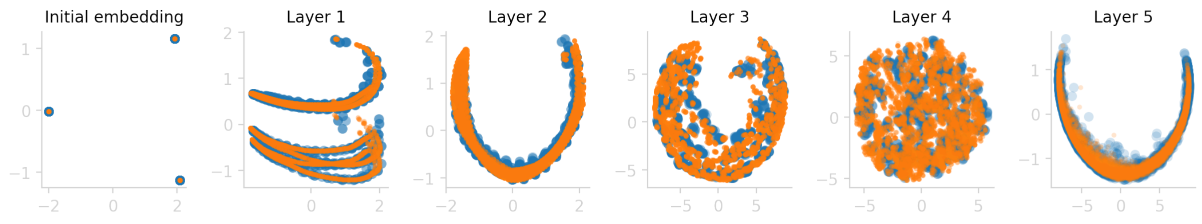


(o) BUCKET SORT (CS)

Figure 4: Performance curves on all tasks for all the positional encodings. The dashed vertical red line is the training range, meaning that sequences to the right have not been seen during training and thus measure length generalization. The sequences to the left of the dashed line visualize the in-domain generalization performance.



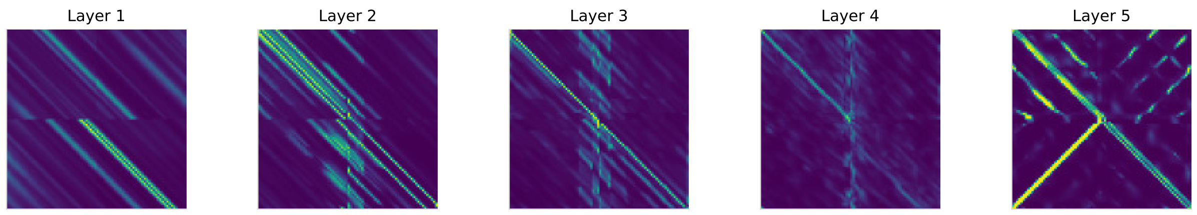
(a) Relative positional encoding (Dai et al., 2019).



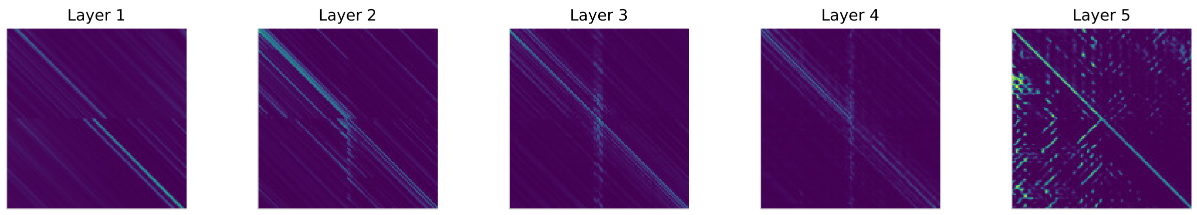
(b) Randomized relative positional encoding (ours).

Figure 5: 2D PCA projections of the activations of the initial embeddings and the encoder layers for 30 sequences on the REVERSE STRING task. For sequence-lengths beyond the training length (shown in orange), the standard relative encoding clearly leads to out-of-distribution activations for layers 3 and 4 compared to those obtained with the maximum training length (shown in blue). In contrast, our randomized version does not lead to out-of-distribution activations for sequences longer than the maximum training length, confirming the intuition in Fig. 1.

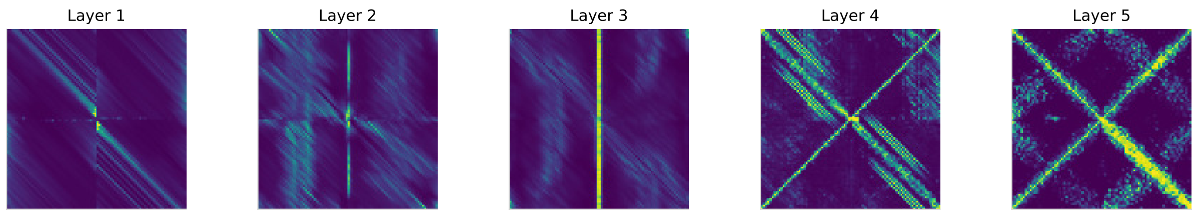
ized version on the REVERSE STRING task. To that end, we follow Csordás et al. (2022) and visualize the maximum over the 8 attention matrices (one per head) for each of the 5 layers in Fig. 6. We compare the attention matrices for sequences of length 40 (i.e., the maximum training length) and 150 (i.e., significantly longer than the maximum training length). For length 40, both encodings produce a noticeable X pattern, which corresponds to the reversal of the string. However, for length 150, the pattern only remains visible for our randomized encodings while it breaks down for the original version, indicating the failure to generalize.



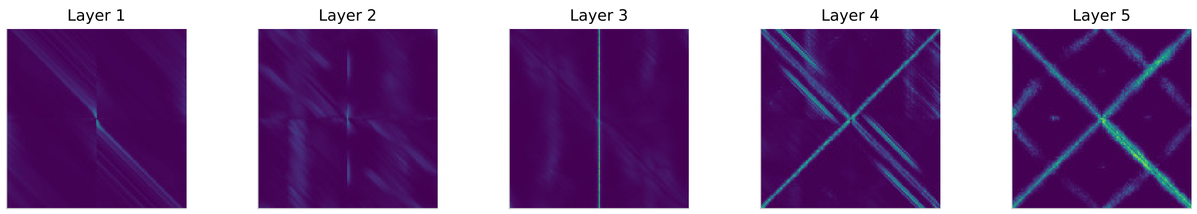
(a) Relative (baseline) with a sequence of length 40 (in-distribution).



(b) Relative (baseline) with a sequence of length 150 (out-of-distribution).



(c) Randomized relative (our method) with a sequence of length 40 (in-distribution).



(d) Randomized relative (our method) with sequence of length 150 (out-of-distribution).

Figure 6: Analysis of the attention matrices for the relative and randomized relative positional encodings on the REVERSE STRING task using sequences of length 40 (i.e., maximum training length) and 150 (i.e., beyond training lengths). We visualize the maximum over the 8 heads per layer (following Csordás et al., 2022) and observe a clear X pattern, which corresponds to the reversal of the sequence. Our randomized relative encodings maintain that pattern on longer sequences, while it breaks down for the standard relative encoding.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Limitations section

☐ A2. Did you discuss any potential risks of your work?
Not applicable. Left blank.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Section 1

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

Section 4 and Appendix A

☑ B1. Did you cite the creators of artifacts you used?
Section 4 and Appendix A

☑ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Appendix A

☑ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Section 4 and Appendix A

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Not applicable. Left blank.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Not applicable. Left blank.

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Section 4 and Appendix A

C ☑ Did you run computational experiments?

Section 4 and Appendix B

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Section 4 and Appendix A

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Section 4 and Appendices A and B

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
Appendix B

☐ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Not applicable. Left blank.

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?
Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.