Recurrent Transformers Trade-off Parallelism for Length Generalization on Regular Languages

Paul Soulos[1,2*]
psoulos1@jhu.edu

Aleksandar Terzić[1,3]
aleksandar.terzic1@ibm.com

Michael Hersche[1]
michael.hersche@ibm.com

Abbas Rahimi[1]
abr@zurich.ibm.com

[1]IBM Research – Zurich, [2]Dept. of Cognitive Science, Johns Hopkins University, [3]ETH Zurich

Abstract

Transformers have achieved remarkable success in Natural Language Processing but struggle with state tracking and algorithmic reasoning tasks, such as modeling Regular Languages. In contrast, Recurrent Neural Networks (RNNs) exhibit perfect generalization modeling Regular Languages. To bridge this gap, we explore Recurrent Transformer variants that incorporate chunking, balancing the parallelizability of Transformers with the sequential processing of RNNs. We identify layer-recurrence as the key type of recurrence that allows Recurrent Transformers to succeed in modeling Regular Languages. Further analysis indicates a rapid decline in generalization performance as chunk size increases beyond two, though with an exponential decrease in training time. This study underscores the critical role of layer-recurrence and chunk size in Recurrent Transformers, highlighting the trade-off between generalization capabilities and parallelism. Code available at `https://github.com/IBM/recurrent-chunked-models-regular-languages`.

1 Introduction

Despite the remarkable success of Transformers [Vaswani et al., 2017] in Natural Language Processing (NLP), the architecture continues to struggle with state tracking and algorithmic reasoning [Hahn, 2020, Bhattamishra et al., 2020, Deletang et al., 2022]. The Chomsky Hierarchy [Chomsky, 1956, 1959] provides a framework for classifying tasks based on difficulty, and Deletang et al. [2022] demonstrated that Transformers show profound weaknesses across the hierarchy.

Regular Languages, the simplest class in the Chomsky Hierarchy, are an important benchmark for evaluating the reasoning capabilities of neural network architectures. Regular Languages can be represented using a Finite State

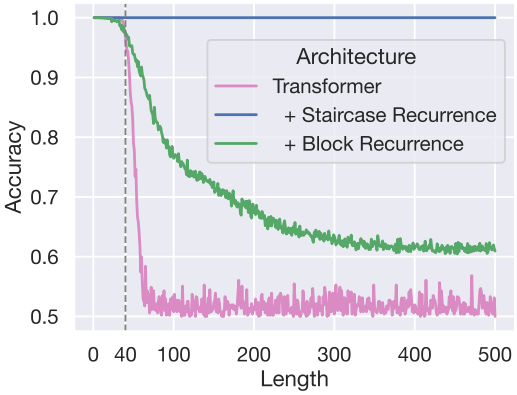

Figure 1: The generalization performance of Transformers with various types of recurrence. Lengths to the left of the dashed line are in-distribution, and lengths to the right are out-of-distribution.

[*]Research conducted at IBM Research – Zurich.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

Machine with no access to an external memory [Hopcroft et al., 2006]. If an architecture fails to learn Regular Languages in a robust and generalizable way, it is unlikely that it will succeed on more challenging problems further up the hierarchy. On three out of four Regular Language tasks from Deletang et al. [2022], Transformers fail to generalize to longer samples than those encountered in the training set (see the pink line in Figure 1). Such experimental studies are supported by theoretical results that highlight the Transformer's inability to model regular languages over arbitrary input lengths [Merrill and Sabharwal, 2023, Strobl et al., 2024].

Understanding the necessary modifications to Transformers to improve their performance modeling Regular Languages is an important step in ascending the hierarchy towards mildly-context sensitive languages, which are hypothesized to encapsulate human language [Joshi, 1985, Joshi et al., 1990]. Furthermore, the modifications might shed light on important inductive biases for human-like language processing leading to improved generalization, reasoning, and sample efficiency.

The performance of Transformers contrasts with the results from various Recurrent Neural Network (RNN) architectures, such as Simple Recurrent Networks [Elman, 1990] and Long Short-Term Memory networks [Hochreiter and Schmidhuber, 1997], both of which exhibit perfect length generalization on the same four Regular Language tasks. RNNs can track intermediate states by updating the recurrent vector that is passed forward to future tokens. For a sequence of T tokens, an RNN will perform T computation steps, allowing it to maintain and update a state representation at each step. In contrast, standard Transformers are non-recurrent, and a Transformer with L layers will process T tokens in L computation steps, regardless of the sequence length. Liu et al. [2022] demonstrated that Transformers with at least $L = \log(T)$ layers can learn shortcut solutions for modeling automata. While these shortcut solutions perform well in-distribution, they are brittle and fail to generalize to longer samples, revealing a limitation in the Transformers' ability to model Regular Languages effectively.

The lack of a recurrent mechanism allows Transformers to process an entire sequence in parallel using masking and teacher forcing. However, the absence of recurrence makes generalization challenging for Transformers on certain algorithmic tasks where maintaining intermediate states would be beneficial [Merrill and Sabharwal, 2023]. In a separate line of work, researchers have investigated combining Transformers with recurrent mechanisms to improve the performance in modeling long sequences [Hutchins et al., 2022, Ding et al., 2021, Bulatov et al., 2022] and tasks where Transformers fail [Ju et al., 2022]. In this paper, we address how Recurrent Transformers affect generalization.

Recurrent Transformer variants introduce the notion of *chunking* to interpolate between the fast, parallelizable architecture of a standard Transformer and the slower, sequential architecture of RNNs. Within a chunk, all tokens are processed in parallel, while a recurrent connection is established between chunks. At the extremes, a chunk size of 1 processes inputs similarly to an RNN, whereas a chunk size equal to the maximum sequence length is equivalent to a standard Transformer. Figure 2 illustrates the computation graph and associated parallelizability for an RNN, a Transformer, and a Recurrent Transformer. It is important to note that the slowdown from adding recurrent processing to Transformers only occurs when teacher forcing is enabled, typically during training. When a standard Transformer generates tokens in an autoregressive setting, it must proceed token-by-token and has no parallelism benefits over a Recurrent Transformer.

We start by defining Token Layer Recurrence, the type of recurrence implemented by RNNs. We show that Staircase Attention [Ju et al., 2022] satisfies Token Layer Recurrence at a chunk size of 1, whereas Block-Recurrent Transformer (BRT) [Hutchins et al., 2022] does not. We validate this finding empirically by showing that Staircase Attention generalizes well on the Regular Language tasks and BRT fails (Figure 1). We also define Chunk Layer Recurrence, a less restrictive version of Token Layer Recurrence, and investigate the relationship between larger chunk sizes and generalization performance. While large chunk sizes are faster and more amenable to large-scale training, we find that small chunk sizes consistently perform better for every architecture in terms of length generalization. Although models with smaller chunk sizes generalize better, we observe that these smaller chunk sizes lead to training speeds an order of magnitude slower than a standard Transformer. This work highlights the importance of either increasing the speed of Recurrent Transformers, finding the optimal chunk size for a specific task, or adding recurrence during post-training.

The primary contributions of this paper are:

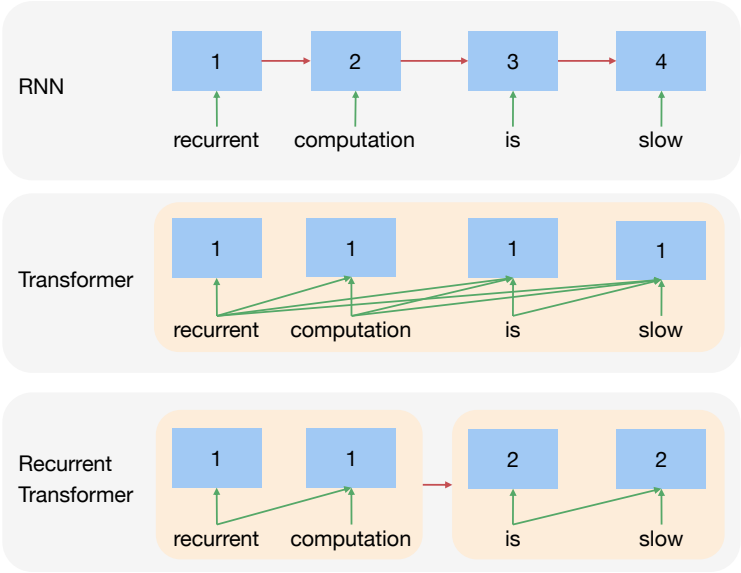

Figure 2: Computation graphs for an RNN, Transformer, and Recurrent Transformer over four tokens. The boxes with the same number can be calculated in parallel, and the number indicates the sequential step of computation (for example, box 3 cannot be calculated until boxes 1 and 2 are finished). Boxes within the same chunk are highlighted in light orange and share the same number. Recurrent connections are red, and attention connections are green. The Recurrent Transformer depicted has a chunk size of 2.

- Defining Token Layer Recurrence and Chunk Layer Recurrence.
- Evaluating three chunk-based models, namely Staircase Attention, Block-Recurrent Transformer, and RegularGPT, on Regular Language tasks.
- Identifying Staircase Attention as a Recurrent Transformer that generalizes well on the tasks and satisfies Chunk Layer Recurrence.
- Investigating the relationship between chunk size and generalization performance.
- Demonstrating the impact of chunk size on training speed and therefore scalability.

2 Related Work

The Chomsky Hierarchy The Chomsky Hierarchy [Chomsky, 1956, 1959] identifies Formal Language computations by increasing complexity based on the required grammar to model the computation. Each level of the hierarchy can be modeled by a finite state machine with access to increasingly expressive data structures. One line of research is to map Neural Network architectures to levels of the hierarchy [Bhattamishra et al., 2020, Deletang et al., 2022, Strobl et al., 2024]. Borenstein et al. [2024] extend previous work on Regular Language recognition to Regular Language modeling by using KL Divergence between model predictions and probabilistic finite-state automata.

Transformers and their limitations Standard Transformers are known to possess computational limitations due to the attention mechanism [Merrill and Sabharwal, 2023, Strobl et al., 2024]. Liu et al. [2022] showed theoretically and empirically that Transformers are able to learn shortcuts to simulate an automata with T tokens in $\log(T)$ layers. The learned shortcuts are brittle and do not generalize well. Chi et al. [2023] modify a Transformer to succeed on Regular Language tasks, although their architecture has degraded performance on natural language modeling.

Recurrent Neural Networks Recurrent Neural Networks (RNNs) were developed to process sequential information, with a particular focus on language processing [Jordan, 1997, Elman, 1990]. Two significant improvements to RNNs were the introduction of gating [Hochreiter and Schmidhuber, 1997, Cho et al., 2014] and attention [Bahdanau et al., 2015]. Researchers have investigated the

computability of RNNs from a formal language perspective [Siegelmann and Sontag, 1995, Chen et al., 2018, Nowak et al., 2023, Svete and Cotterell, 2023].

Recurrent Transformers Prior work has investigated adding recurrent mechanisms into Transformers. Ju et al. [2022] introduce Staircase Attention and show improvements on entity-tracking and language modeling. Hutchins et al. [2022] and Didolkar et al. [2022] contemporaneously published Transformer architectures which use cross attention to a set of recurrent vectors to improve modeling long range sequences. Bulatov et al. [2022] showed improvements on long range modeling by adding recurrent memory tokens to Transformers. Chowdhury and Caragea [2024] investigated the relationship between Transformer depth-wise recurrence and chunk-wise recurrence on various tasks.

Transformer Length Generalization Since Transformers lack recurrence, positional embeddings are used to signify the order of tokens. Different types of positional embeddings can improve length generalization [Csordás et al., 2021, Ruoss et al., 2023, Shaw et al., 2018, Su et al., 2024, Zhou et al., 2024, Kazemnejad et al., 2023, Press et al., 2021], although Deletang et al. [2022] and Ruoss et al. [2023] showed that none of the positional embeddings that they tested allowed Transformers to generalize well on Regular Languages.

3 Architectures

We briefly describe three architectures that we test. For additional details, please see the original papers. For each architectures, task, and chunk size, we run a hyperparameter search to find the best setting for that combination. We then train five randomly initialized models with the best setting and report these results. Training details for each architecture can be found in Appendix A.

The three architectures we discuss below use the notion of dividing an input sequence $X = \{X_0, \dots X_{T-1}\}$ of length T into chunks $K = \{K_0, \dots K_{\lceil T/C \rceil}\}$ of size C where $K_i = \{X_{i \times C}, \dots X_{(i+1) \times C - 1}\}$. Each architecture processes chunks in different ways.

Token Layer Recurrence RNNs implement recurrence through the property that the output for a token $X_{i,l}$ at sequence position i and layer l depends on the output of a previous token at layer l: $X_{i,l} = f(X_{<i,l}, \dots)\ \forall i > 0$. Meanwhile, the state of token $X_{i,l}$ in a Transformer with causal attention depends only on states at $l-1$: $X_{i,l} = f(\{X_{<i,l-1}\})$. Token layer recurrence allows a network to iteratively update a state as new information is processed. For a treatment of token layer recurrence and generalization in Linear RNNs, see Fan et al. [2024] and Merrill et al. [2024].

Chunk Layer Recurrence When the chunk size is greater than 1, the property of layer recurrence is violated. Consider a chunk size of $C = 2$. X_0 and X_1 will be processed in parallel, which means that $X_{1,l} = f(X_{0,l-1}, X_{1,l-1})$. Thus, we define Chunk Layer Recurrence similarly to Token Layer Recurrence but with chunks: $K_{i,l} = f(K_{<i,l}, \dots)\ \forall i > 0$. A model that satisfies Chunk Layer Recurrence will satisfy Token Layer Recurrence when the chunk size is set to 1.

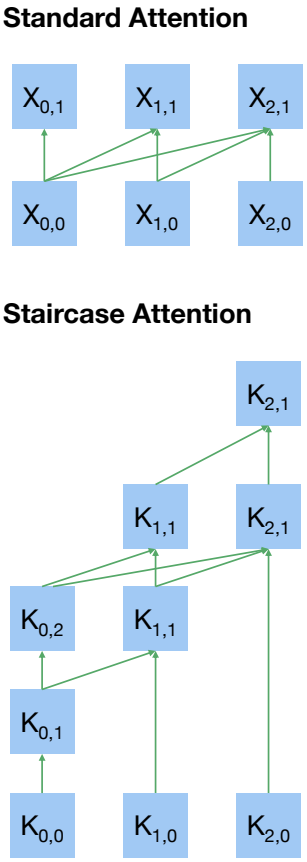

Figure 3: Attention patterns for standard attention and Staircase Attention.

3.1 Staircase Attention

Staircase Attention [Ju et al., 2022] staggers the chunks so that each consecutive chunk attends to the previous chunk after one level of processing. The differences between standard attention and Staircase Attention are depicted in Figure 3. By staggering the chunks, Staircase Attention satisfies Chunk Layer Recurrence. In order to speed up the model, Staircase Attention uses a Universal

Transformer [Dehghani et al., 2018] so that multiple chunks at different layers can be processed in parallel. Ju et al. [2022] show that Staircase Attention succeeds on two state tracking tasks where Transformers fail, but they did not consider how the model generalizes out-of-distribution. Here, we look at state tracking tasks and how well the model generalizes to longer sequences. Staircase Attention also shows competitive performance on language modeling tasks. The authors consider chunk sizes 8, 16, and 32 for the algorithmic tasks, and chunk sizes 32, 64, and 128 for the language modeling tasks. Their results generally show improved performance with smaller chunk sizes, at the cost of a slower model.

3.2 Block-Recurrent Transformers

Block-Recurrent Transformers (BRT) [Hutchins et al., 2022] integrate the recurrent connection as cross attention between the current chunk and a set of recurrent tokens. This allows the output of the chunk to use information from previous chunks. The recurrent tokens are updated by using cross-attention to the chunk tokens, and the recurrent tokens are passed along to the next chunk. Crucially, the recurrent tokens attend to the input of the chunk instead of the output. This means that BRT *does not* satisfy Chunk Layer Recurrence. Inspired by gating in LSTMs [Hochreiter and Schmidhuber, 1997], BRT experiments with different gate configurations for the recurrent tokens. This design is similar to the Temporal Latent Bottleneck [Didolkar et al., 2022] which was published concurrently. BRT shows improved performance on long sequence language modeling as measured by perplexity. They uses a chunk size of 512, two orders of magnitude larger than the chunks we consider in this paper.

3.3 RegularGPT

RegularGPT [Chi et al., 2023] is not a recurrent model, but we include it in our experiments since it uses chunked processing. Rather than enforcing a sequential processing of chunks, RegularGPT leverages attention masking to enforce a divide-and-conquer strategy. RegularGPT does not support chunk size 1. The divide-and-conquer strategy is implemented using a Universal Transformer with sliding dilated attention masks and dynamic depth. RegularGPT has good length generalization on the Regular Language tasks from [Deletang et al., 2022] using a chunk size of $C = 2$. However, while RegularGPT improves performance on the Regular Language tasks, it leads to worse performance in Language Modeling as measured by perplexity on length extrapolation when compared to other models designed for long contexts. RegularGPT shows decreased performance going from chunk size 2 to 3. Here, we investigate whether this trend continues for larger chunk sizes 4 and 8. On language modeling, RegularGPT is tested with chunk sizes 32, 64, 128, and 256. The authors find that chunk size 128 works the best but underperforms baselines.

4 Results

4.1 Datasets

We test the three models on the four Regular Language tasks from Deletang et al. [2022]: Even Pairs, Parity Check, Cycle Navigation, and Modular Arithmetic (Simple). For more details about these tasks, see Appendix B. We follow the same procedure as Deletang et al. [2022] and train our models on sequences up to length 40, and during test time we evaluate our model on sequences up to length 500. The score is the average accuracy measured over sequences from length 1 to 500.

4.2 Token Recurrent Transformers (Chunk Size 1)

We first test Recurrent Transformers with a chunk size of 1 to see if a token recurrent model

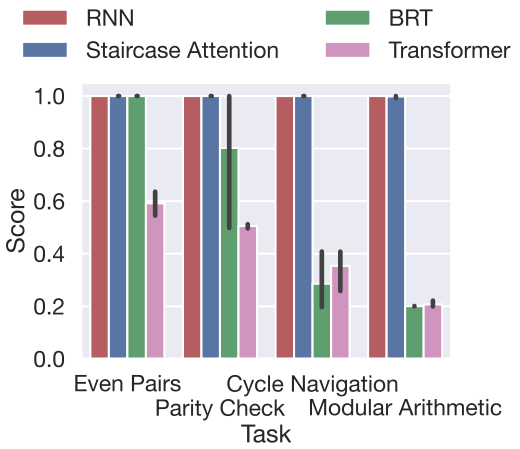

Figure 4: Average generalization performance of token recurrent architectures (chunk size 1) on Regular Language tasks. Bars indicate minimum and maximum values across five random seeds.

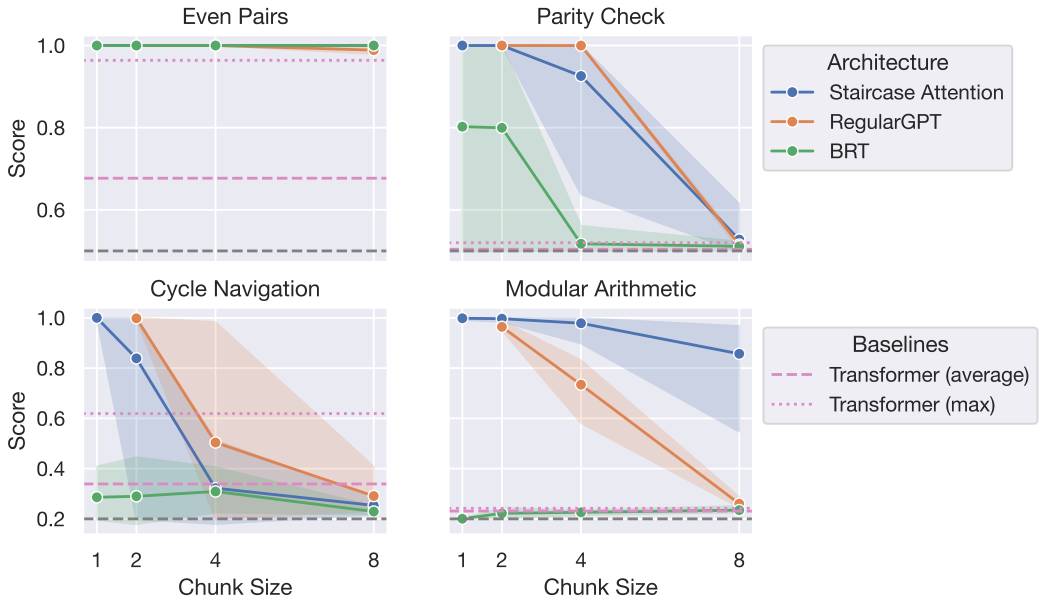

Figure 5: Average generalization performance by chunk size across five random seeds. The shaded regions represent minimum and maximum values. The dashed gray bar represents chance performance. The dashed pink line represents the average Transformer performance, and the dotted pink link represents the max Transformer performance. Transformer results are drawn from Deletang et al. [2022].

can replicate the performance of a simple RNN. The results are shown in Figure 4. Both the RNN and Staircase Attention achieve perfect generalization across the four tasks. BRT performs better than the Transformer in Parity Check and Even Pairs but does not provide any benefit in Cycle Navigation or Modular Arithmetic. These results validate that adding the right type of recurrence to Transformers can improve generalization, but a chunk size of 1 forgoes the parallelism benefits of Transformers.

4.3 Performance degrades with larger chunk sizes

In order to see how much we can scale up Recurrent Transformers, we test how the generalization performance changes as we increase parallelism through larger chunk sizes. Figure 5 shows the performance of each architecture and chunk size combination on the four tasks. Across all of the tasks and architectures, generalization drops as chunk sizes increases, except for Even Pairs where all models succeed.

Even Pairs Even Pairs is the only Regular Language task where a standard Transformer is capable of generalizing well, although with high variance as seen by the difference between max and average Transformer results indicated by the pink lines. All three models achieve consistent generalization on Even Pairs regardless of the chunk size.

Parity Check Transformers struggle on Parity Check with the best performing Transformer barely performing above chance. Both Recurrent models are capable of perfect generalization at chunk size 1, and all three chunked models succeed at chunk size 2, although BRT shows high variance. At chunk size 4, BRT fails to generalize, RegularGPT succeeds across every seed, and Staircase Attention generally performs well.

Cycle Navigation After Even Pairs, Cycle Navigation is the second easiest Regular Language task, with the best Transformer achieving a score of 0.6. For chunk sizes 1 and 2, Staircase Attention generalizes well, but by chunk size 4 the performance drops drastically. RegularGPT does well at chunk size 2, and is capable of succeeding at chunk size 4, although with high variance. By chunk size 8, RegularGPT no longer generalizes. BRT fails to generalize at any chunk size.

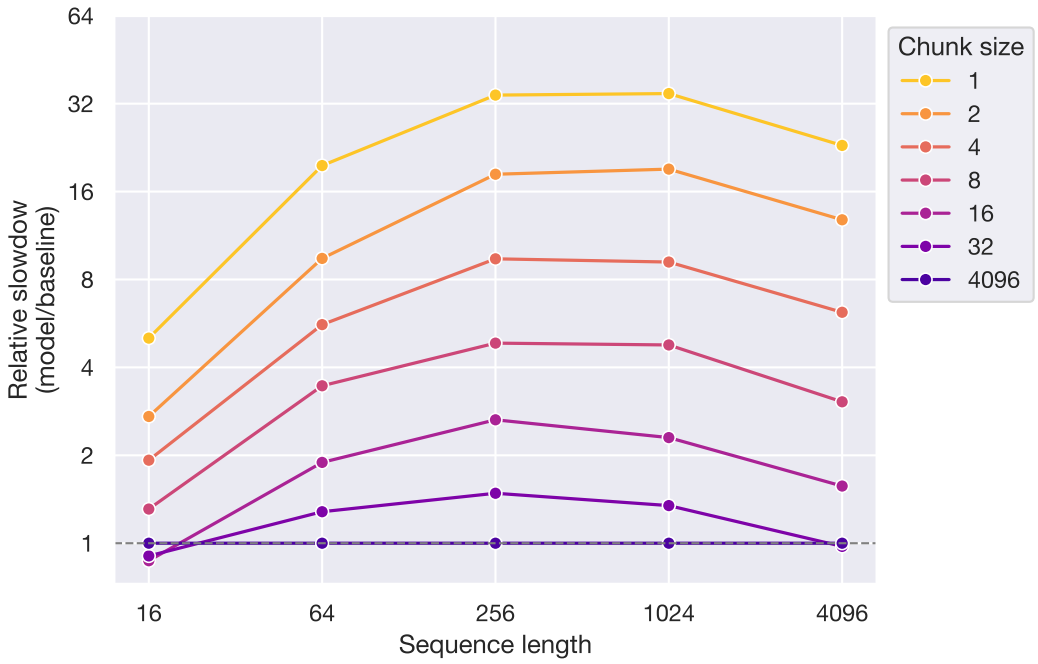

Figure 6: Staircase Attention relative step time slowdown compared to a baseline Transformer, represented by the dashed line. Relative slowdown is the ratio of model step time by baseline step time.

Modular Arithmetic Transformers struggle on Modular Arithmetic, and the best performing model barely does better than chance. BRT fails to generalize and shows similar results as the Transformer. RegularGPT does well at chunk size 2, but it's performance decreases linearly down to chance by chunk size 8. Staircase Attention does well across all of the chunk sizes tested.

The main takeaway as evidenced by Figure 5 is that smaller chunk sizes work better. Staircase Attention with chunk size 2 consistently performs well. We do not find much difference between chunk size 1 and 2 in terms of generalization performance for Staircase Attention, and as we will show in the next section, chunk size 2 is roughly twice as fast as chunk size 1.

4.4 Training Speed

The previous section showed that smaller chunk sizes for Recurrent Transformers work better. However, smaller chunk sizes also lead to a slower step time when the architectures are trained in a non-autoregressive setting with teacher forcing. In this section, we quantify how much slower Recurrent Transformers are at different chunk sizes.

Figure 6 shows the relative step time slowdown of Staircase Attention for different chunk sizes and sequence lengths. We use relative speed in order to profile the model speed since relative speed transfers across hardware and implementation details. In order to calculate the relative speed, we divide the forward and backward step time for Staircase Attention by the step time for a parallel Transformer. When a chunk size is greater than or equal to the sequence length, the performance should mirror that of a standard Transformer since the sequence will be processed without recurrence, which we see for chunk size 4096. The figure shows that doubling the chunk size roughly makes the model twice as fast. As the sequence length increases so does the slowdown, until a certain point, after which the slowdown decreases. Staircase Attention does not compute the full $O(N^2)$ attention, so at larger lengths the slowdown compared baseline starts to recede. The relative step slowdown for BRT is shown in Appendix C.

Importantly, at test time when performing autoregressive generation, Recurrent Transformers do not incur a speed penalty for their recurrence. This means that these architectures are a fixed training cost, and they yield the same cost when generating text in an autoregressive manner.

4.5 Navigating the generalization-parallelism frontier

The previous two sections illustrate the challenge of scaling Recurrent Transformers. On one hand, large chunk sizes lead to faster training, whereas on the other hand, smaller chunk sizes are more consistent across different tasks. Ideally, we would have a theoretical basis for identifying optimal chunk size in a natural language corpus. Even on the four simple tasks explored here, Staircase Attention has a different optimal chunk size for each task. While chunk size 2 generalizes well on every task, Modular Arithmetic is amenable to chunk size 8 which is roughly 3 times faster than than chunk size 2.

5 Conclusion and Future Work

We investigated the performance of Recurrent Transformers on Regular Language tasks. While Transformers generalize poorly on these tasks, we found that Staircase Attention with a small chunk size generalizes well.

The small chunk size required for consistent generalization poses some roadblocks for scaling up Staircase Attention. Two immediate questions are whether we can define the optimal chunk size that supports generalization for a given task, and whether we can instill recurrence during post-training instead of pre-training.

In order to scale up Recurrent Transformers for reasoning, we want to find the largest chunk size for a dataset that retains good generalization. For instance, Python is whitespace sensitive, so one trivial chunking hypothesis is to break chunks on newlines. A similar idea is to break natural language on punctuation, conjunctions, and relative clauses. Future work should investigate the properties that define optimal chunks.

One way to alleviate the slowdown from recurrent training is to start with a pre-trained LLM and instill recurrence during post-training. Recurrent processing is more likely to be helpful on datasets where reasoning is important. For example, recurrence would be much more beneficial on datasets with explicit intermediate reasoning [Nye et al., 2021, Wei et al., 2022] than a dataset with a list of independent facts. Some forms of post-training are inherently autoregressive like reinforcement learning from human feedback [Christiano et al., 2017], process supervision [Lightman et al., 2024], and preference optimization [Pang et al., 2024]; in these cases there will be no extra cost of using Recurrent Transformers over standard Transformers. Future work should investigate how recurrent post-training can be performed on a model pre-trained in a parallel manner.

An additional direction to consider is how Recurrent Transformers perform within the scaling laws [Kaplan et al., 2020, Hoffmann et al., 2022]. If recurrence is a useful inductive bias, models may converge faster with better generalization while being trained on shorter sequences. This might make Recurrent Transformers more competitive when evaluated on computational budget as opposed to only looking at the time for a training step.

This work showed that some Recurrent Transformers can successfully model Regular Languages. Human language is hypothesized to be Mildly Context-Sensitive [Joshi, 1985, Joshi et al., 1990], and we can climb the Chomsky Hierarchy [Chomsky, 1956, 1959] by integrating Recurrent Transformers with differentiable stacks [Joulin and Mikolov, 2015, Grefenstette et al., 2015] to model Context-Free Language and differentiable trees [Soulos et al., 2023] to model Mildly Context-Sensitive Languages. Recent work by DuSell and Chiang [2024] showed that Transformers with Stack Attention improve modeling of Context-Free transformations, but their models do not perform well out-of-distribution. Integrating Staircase Attention into their model may help Transformers perform better on Context-Free transformations in and out-of distribution.

References

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017. arXiv: 1706.03762.

Michael Hahn. Theoretical Limitations of Self-Attention in Neural Sequence Models. *Transactions of the Association for Computational Linguistics*, 8:156–171, 2020. Place: Cambridge, MA Publisher:

MIT Press.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the Ability and Limitations of Transformers to Recognize Formal Languages. In , *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, Online, November 2020. Association for Computational Linguistics.

Gregoire Deletang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A. Ortega. Neural Networks and the Chomsky Hierarchy. In *The Eleventh International Conference on Learning Representations*, September 2022.

N. Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

Noam Chomsky. On Certain Formal Properties of Grammars. *Inf. Control.*, 2:137–167, 1959.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA, June 2006.

William Merrill and Ashish Sabharwal. The Parallelism Tradeoff: Limitations of Log-Precision Transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023. Place: Cambridge, MA Publisher: MIT Press.

Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What Formal Languages Can Transformers Express? A Survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, May 2024.

Aravind K Joshi. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? *Natural Language Parsing*, pages 206–250, 1985. Publisher: Cambridge University Press.

Aravind K Joshi, Krishnamurti Vijay-Shanker, David Weir, and others. The convergence of mildly context-sensitive grammar formalisms. 1990. Publisher: Citeseer.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, April 1990.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8): 1735–1780, November 1997.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers Learn Shortcuts to Automata. In *The Eleventh International Conference on Learning Representations*, September 2022.

DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. Block-Recurrent Transformers. In , *Advances in Neural Information Processing Systems*, 2022.

SiYu Ding, Junyuan Shang, Shuohuan Wang, Yu Sun, Hao Tian, Hua Wu, and Haifeng Wang. ERNIE-Doc: A Retrospective Long-Document Modeling Transformer. 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 2914–2927, Online, August 2021. Association for Computational Linguistics.

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent Memory Transformer. *Advances in Neural Information Processing Systems*, 35:11079–11091, December 2022.

Da Ju, Stephen Roller, Sainbayar Sukhbaatar, and Jason E. Weston. Staircase Attention for Recurrent Processing of Sequences. In , *Advances in Neural Information Processing Systems*, October 2022.

Nadav Borenstein, Anej Svete, Robin Chan, Josef Valvoda, Franz Nowak, Isabelle Augenstein, Eleanor Chodroff, and Ryan Cotterell. What Languages are Easy to Language-Model? A Perspective from Learning Probabilistic Regular Languages. In , *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15115–15134, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

Ta-Chung Chi, Ting-Han Fan, Alexander Rudnicky, and Peter Ramadge. Transformer Working Memory Enables Regular Language Reasoning And Natural Language Length Extrapolation. In , *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5972–5984, Singapore, December 2023. Association for Computational Linguistics.

Michael I Jordan. Serial order: A parallel distributed processing approach. In *Advances in psychology*, volume 121, pages 471–495. Elsevier, 1997.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *EMNLP*, 2014.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In , *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

H. T. Siegelmann and E. D. Sontag. On the Computational Power of Neural Nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995.

Yining Chen, Sorcha Gilroy, Andreas Maletti, Jonathan May, and Kevin Knight. Recurrent Neural Networks as Weighted Language Recognizers. In , *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2261–2271, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Franz Nowak, Anej Svete, Li Du, and Ryan Cotterell. On the Representational Capacity of Recurrent Neural Language Models. In , *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7011–7034, Singapore, December 2023. Association for Computational Linguistics.

Anej Svete and Ryan Cotterell. Recurrent Neural Language Models as Probabilistic Finite-state Automata. In , *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8069–8086, Singapore, December 2023. Association for Computational Linguistics.

Aniket Rajiv Didolkar, Kshitij Gupta, Anirudh Goyal, Nitesh Bharadwaj Gundavarapu, Alex Lamb, Nan Rosemary Ke, and Yoshua Bengio. Temporal Latent Bottleneck: Synthesis of Fast and Slow Processing Mechanisms in Sequence Learning. In *Advances in Neural Information Processing Systems*, October 2022.

Jishnu Ray Chowdhury and Cornelia Caragea. Investigating Recurrent Transformers with Dynamic Halt, March 2024. arXiv:2402.00976 [cs].

Róbert Csordás, Kazuki Irie, and Juergen Schmidhuber. 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*, pages 619–634, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

Anian Ruoss, Grégoire Delétang, Tim Genewein, Jordi Grau-Moya, Róbert Csordás, Mehdi Bennani, Shane Legg, and Joel Veness. Randomized Positional Encodings Boost Length Generalization of Transformers. In , *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1889–1903, Toronto, Canada, July 2023. Association for Computational Linguistics.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-Attention with Relative Position Representations. In , *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. Publisher: Elsevier.

Yongchao Zhou, Uri Alon, Xinyun Chen, Xuezhi Wang, Rishabh Agarwal, and Denny Zhou. Transformers Can Achieve Length Generalization But Not Robustly, February 2024. arXiv:2402.09371 [cs].

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The Impact of Positional Encoding on Length Generalization in Transformers. In , *Advances in Neural Information Processing Systems*, volume 36, pages 24892–24928. Curran Associates, Inc., 2023.

Ofir Press, Noah Smith, and Mike Lewis. Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. In *International Conference on Learning Representations*, October 2021.

Ting-Han Fan, Ta-Chung Chi, and Alexander Rudnicky. Advancing Regular Language Reasoning in Linear Recurrent Neural Networks. In , *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pages 45–53, Mexico City, Mexico, June 2024. Association for Computational Linguistics.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In *Forty-first International Conference on Machine Learning*, April 2024.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal Transformers. In *International Conference on Learning Representations*, September 2018.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, and others. Show Your Work: Scratchpads for Intermediate Computation with Language Models. In *Deep Learning for Code Workshop*, 2021.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain of Thought Prompting Elicits Reasoning in Large Language Models. In , *Advances in Neural Information Processing Systems*, 2022.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep Reinforcement Learning from Human Preferences. *Advances in Neural Information Processing Systems*, 30, 2017.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's Verify Step by Step. In *The Twelfth International Conference on Learning Representations*, 2024.

Richard Yuanzhe Pang, Weizhe Yuan, Kyunghyun Cho, He He, Sainbayar Sukhbaatar, and Jason Weston. Iterative Reasoning Preference Optimization, June 2024. arXiv:2404.19733 [cs].

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling Laws for Neural Language Models. 2020. arXiv: 2001.08361.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Thomas Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karén Simonyan, Erich Elsen, Oriol Vinyals, Jack Rae, and Laurent Sifre. Training compute-optimal large language models. In , *Advances in Neural Information Processing Systems*, volume 35, pages 30016–30030. Curran Associates, Inc., 2022.

Armand Joulin and Tomas Mikolov. Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets. In , *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to Transduce with Unbounded Memory. In *NIPS*, 2015.

Paul Soulos, Edward J Hu, Kate McCurdy, Yunmo Chen, Roland Fernandez, Paul Smolensky, and Jianfeng Gao. Differentiable tree operations promote compositional generalization. In *International Conference on Machine Learning*, pages 32499–32520. PMLR, 2023.

Brian DuSell and David Chiang. Stack Attention: Improving the Ability of Transformers to Model Hierarchical Patterns. In *The Twelfth International Conference on Learning Representations*, 2024.

A Training details

The training details are available below. For each model, we run a hyperparameter search of 20 runs for each combination of architecture, chunk size, and task. The best hyperparameter setting resulting in the highest generalization performance is used to train five additional runs for final results. We noticed that in some cases, models with lower validation loss or higher validation accuracy would have worse generalization performance. When multiple models had the same generalization performance, we selected the setting with the best validation accuracy, and then the lowest validation loss. All models are trained with the AdamW optimizer.

For Staircase Attention, BRT, and RegularGPT, we remove the final EOS token that Deletang et al. [2022] include with the input sequence. For the Transformer model, we keep the final EOS token in order to match their training procedure.

A.1 Staircase Attention

We base our implementation of Staircase Attention on Ju et al. [2022]. Since the model includes recurrence, we do not include positional embeddings. This corresponds to the NoPe scheme [Kazemnejad et al., 2023].

For sweeps of chunk sizes 1 and 2, we train for 20k steps to reduce the amount of compute used. For chunk sizes 4 and 8, we train for 100k steps. For chunk size 8, we noticed that the training and validation accuracies were still rising at 100k steps, so we increased the training steps to 1 million for the final give runs. For Cycle Navigation and chunk size 2, 20k training steps was insufficient so we increased the number of training steps to 100k. We only run the hyperparameter search for 5 runs on Even Pairs since the task is easy.

We sample the following hyperparameter values during our search:

Batch size: [64, 128, 256]
Embedding size: [64, 256, 512]
Number of heads: [4, 8, 16]
Learning rate: log_uniform(1e-5, 1e-3)
Dropout: [0, .1, .2]
Weight decay: [0, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1]
AdamW beta1 inverse: log_uniform(.01, .2)
Number of recurrent calls: [2, 4]
Number of layers in the Universal Transformer: [1, 2]

A.2 RegularGPT

We use the RegularGPT implemention from the authors of Chi et al. [2023] available at `https://github.com/chijames/regular_gpt`.

Unless noted, each search and final run trains for 50k steps. For Cycle Navigation and chunk sizes 4 and 8, we train the final five runs for 100k steps. For Modular Arithmetic, every chunk size gets 100k training steps for the final five runs. On Even Pairs, we only perform a hyperparameter search with 10 runs.

We sample the following hyperparameter values during our search:

Batch size: [64, 128, 256]
Embedding size: [64, 256, 512]
Number of heads: [4, 8, 16]
Learning rate: log_uniform(1e-5, 1e-3)

Dropout: [0, .1, .2]
Weight decay: [0, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1]
AdamW beta1 inverse: log_uniform(.01, .2)
Number of recurrent calls: [2, 4]
Number of layers in the Universal Transformer: [1, 2, 3, 4]

A.3 Block-Recurrent Transformer

We use the Block-Recurrent Transformer (BRT) implementation at `github.com/lucidrains/block-recurrent-transformer-pytorch`. Hutchins et al. [2022] experiment with various gate configurations, and we use select the best performing configuration from the paper: fixed gate with the skip configuration which corresponds to `Rec:fixed:skip` in the original paper.

We use a BRT architecture with three layers, where the first and third layers are standard Transformer layers, and the middle layer is the recurrent layer. We dynamically set the number of recurrent vectors based on the chunk size in order to allow the hyperparameter search to select how much bias there should be between parallel and recurrent computation.

We train runs with chunk size 1 for 50k steps, and runs with chunk sizes 2, 4, and 8 for 100k steps. For the final five runs, we increase the number of training steps for chunk sizes 4 and 8 to 200k. On Cycle Navigation and Modular Arithmetic with chunk size 1, we increase the number of training steps to 100k. We only run the hyperparameter search for 5 runs on Even Pairs since the task is easy.

We sample the following hyperparameter values during our search:

Batch size: [64, 128, 256]
Embedding size: [64, 256, 512]
Number of heads: [4, 8, 16]
Learning rate: log_uniform(1e-5, 1e-3)
Dropout: [0, .1, .2]
Weight decay: [0, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1]
AdamW beta1 inverse: log_uniform(.01, .2)
Number of recurrent vectors: [1x chunk size, 2x chunk size, 4x chunk size]

A.4 Transformer

Unless specified, we follow the hyperparameters from Deletang et al. [2022]. We train for 100k steps because Table B.6 in Deletang et al. [2022] showed that more steps does not help, except for Modular Arithmetic where we increase the number of steps to 1 million. We use causal attention masking with no positional embeddings.

B Dataset details

Parity Check: Parity Check corresponds to a finite-state machine with two states, *Even* and *Odd*, and it admits two inputs, 0 and 1. Given a binary input sequence, the final state of the machine is *Even* if the number of ones in the sequence is even, and otherwise it is *Odd*.

Even Pairs: Given a binary input sequence, the *Even Pairs* task consists of determining whether the number of 01 and 10 subsequences in the input is equal. This task can be equivalently expressed as detecting whether the start and the end of the binary input string are the same symbol.

Cycle Navigation: This task corresponds to a finite-state machine consisting of five states arranged on a cycle graph. It admits three inputs, *Left*, *Right* and *Stay*. Starting from an initial state, the cycle is traversed according to the inputs.

Modular Arithmetic: The inputs to this task are integers between 0 and 4 interleaved with the operations $+$, $-$ and $*$. The final state of the automaton is obtained by evaluating the given expression in modular arithmetic with modulo 5.

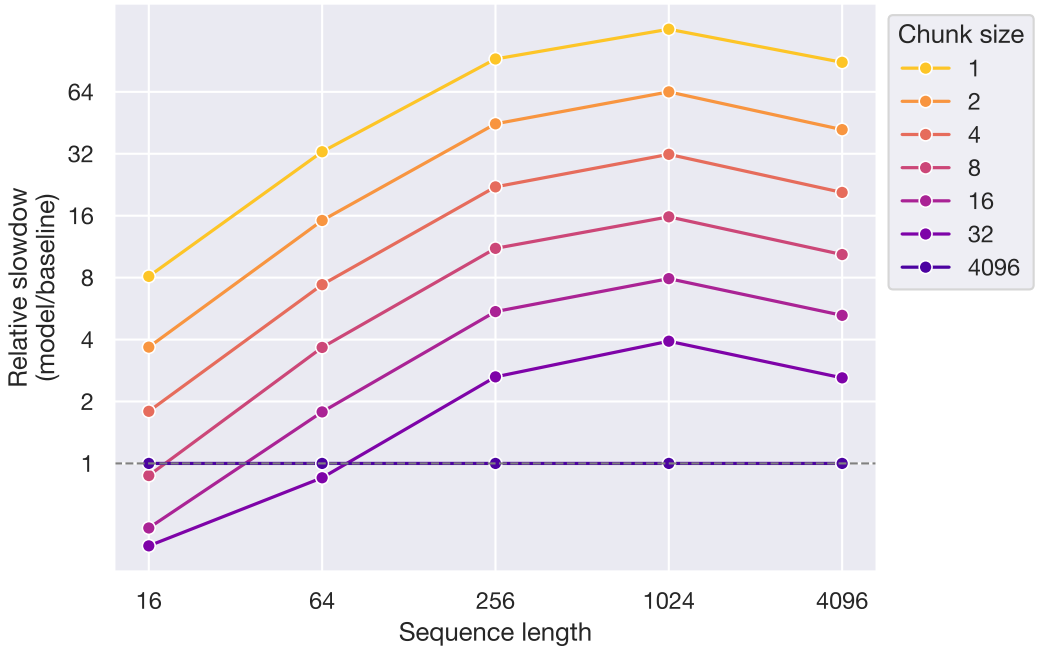

Figure 7: BRT relative step time slowdown compared to a baseline Transformer, represented by the dashed line. Relative slowdown is the ratio of model step time by baseline step time.

C BRT Relative Performance

The training speed analaysis for Staircase Attention in Section 4.4 is replicated here for BRT. The results are shown in Figure 7.