Recurrent Transformers

B.N. Kausik[1]

June 23 2025

Abstract

We propose recurrent transformers, a neural network architecture that combines the simplicity and efficiency of recurrent networks with the efficacy of transformers. Specifically, a recurrent transformer is a transformer that is progressively applied to a fixed-width sliding window across the input sequence, thereby operating in linear time in the length of the sequence during both training and inference. We present three variants (1) **unit recurrence with forget**, which operates on unit-width segments of the input sequence, progressively forgetting information, and suitable for natural language processing; (2) **block recurrence with forget**, which operates on blocks of the input sequence, progressively forgetting information, and suitable for processing multi-dimensional datasets; and (3) **block recurrence with accumulate**, which operates on blocks of the input sequence, progressively accumulating information, suitable for copy and selective copy tasks. While recurrent transformers are inherently sequential, they are memory-efficient and thereby amenable to parallel processing in large batches. In our experiments across each of these variants, we find that a single layer recurrent transformer can match the performance of multi-layer transformers on benchmark datasets, delivering comparable accuracy at the same training cost, but at vastly reduced inference cost.

[1]Author: https://sites.google.com/view/bnkausik/ Contact: bnkausik@gmail.com
Thanks to F. Baskett, J. Jawahar, G. Papadopoulos and P. Tadepalli for suggestions

Introduction

Transformers, Vaswani et al (2017), have found wide use in learning and modeling human perception. Since causal attention, the core computation in a transformer, is quadratic in the length of the input, a broad range of approaches have been proposed to reduce the time and space complexity of transformers in practical applications. The approaches range across model pruning e.g., LeCun et al., (1989), Hassibi et al, (1993) Han et al, (2015), Liu et al, (2019), Blalock et al, (2020), Heoffler et al (2022), Sun et al (2023), and Frantar & Alistarh (2023); model distillation, e.g., Hinton et al, (2015), Chen et al (2017) and Asami et al (2017); modified gradient descent, Kausik (2024); as well as algorithms that directly reduce the computational complexity of calculating attention, e.g., Parmar et al., (2018) Child et al., (2019), Beltagy et al. (2020), Kitaev et al. (2020), Tay et al., (2020), Wang et al., (2020), Bello et al. (2021) Choromanski et al., (2021), Peng et al., (2021), Xiong et al., (2021), Ma et al., (2021), Zheng et al., (2022), Alman and Song, (2023), Ma et al, (2023), Han et al., (2024), Ma et al., (2024), and Kannan et al (2024).

Preceding transformers, Recurrent Neural Networks (RNNs), e.g., Elman (1990), their variants Long Short-Term Memory (LSTM), e.g., Hochreiter & Schmidhuber (1997) and Gated Recurrent Units (GRUs), e.g., Cho et al., (2014), were popular for linear time processing of sequential data streams such as natural language processing. More recent architectural alternatives include State Space Models, e.g., Gu et al (2021), and minimal RNNs, Feng et al (2024).

Building on prior work, we propose recurrent transformers, combining the simplicity and efficiency of recurrent networks with the efficacy of transformers. Specifically, a recurrent transformer is a transformer that is progressively applied to a fixed-width sliding window across the input sequence, thereby operating in linear time in the length of the sequence during both training and inference. We present three variants (1) **unit recurrence with forget**, which operates on unit-width segments of the input sequence, progressively forgetting information, and suitable for natural language processing; (2) **block recurrence with forget**, which operates on blocks of the input sequence, progressively forgetting information, and suitable for processing multi-dimensional datasets; and (3) **block recurrence with accumulate**, which operates on blocks of the input sequence, progressively accumulating information, suitable for copy and selective copy tasks. While recurrent transformers are inherently sequential, they are memory-efficient and thereby amenable to parallel processing in large batches.

In our experiments across each of these variants, we find that a single layer recurrent transformer can meet or beat the performance of multi-layer transformers on benchmark datasets, delivering comparable accuracy at the same training cost, but at vastly reduced inference cost.

Architecture

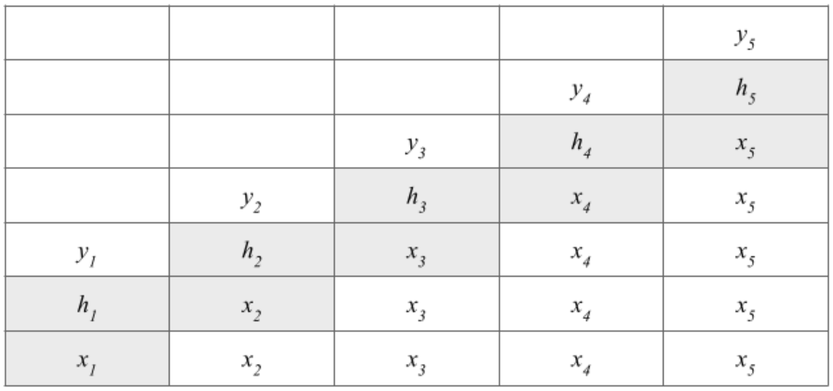
Unit recurrence with forget

Let $X = (x_1, \dots x_i, \dots x_N)$ be an input sequence, where each x_i is a token represented as a vector embedding. Let T denote the function computed by a transformer in that on input sequence X the transformer outputs $T(X) = (y_1, y_2, \dots y_N)$. We assume that the x_i and y_i are vectors of the same dimension, and that T preserves causality in that y_i does not depend on x_{i+1}, x_{i+2},..., x_N. The recurrent transformer based on T is as follows:

$$h_1 = T(x_1) \qquad (1)$$
$$(y_i, h_{i+1}) = T(h_i, x_{i+1}) \qquad (2)$$
$$y_N = T(h_N) \qquad (3)$$

In the above, h_i and y_i are the hidden state and output at position i respectively. Equation (1) initializes the recurrence at position 1, Equation (2) iteratively applies the transformer function with causal attention between position $i + 1$ and position i, and Equation (3) closes the recurrence at the last position.

Fig. 1 is a visual representation of the recurrent transformer, where the bottom row of the table shows an input sequence x_1, x_2, x_3, x_4 of five tokens. Per Equation (1), the transformer function T is first applied to input x_1 to produce the hidden state h_1. Then per Equation (2), the transformer function is applied to the sequence (h_1, x_2) to obtain (y_1, h_2). At each step i, h_{i-1} is forgotten. And so on, capping the recurrence per Equation (3) at the last position.



				y_5
			y_4	h_5
		y_3	h_4	x_5
	y_2	h_3	x_4	x_5
y_1	h_2	x_3	x_4	x_5
h_1	x_2	x_3	x_4	x_5
x_1	x_2	x_3	x_4	x_5

Fig. 1: Visual representation of the recurrent transformer architecture. Each shaded box represents an application of the underlying transformer layer function to produce the token(s) in the box(es) above.

Block recurrence with forget

A block recurrence transformer is similar to a unit-recurrence transform but simply operates on non-overlapping sequential blocks of multiple tokens at each step. Specifically, in the input sequence $(x_1,... x_i, ... x_N)$, each x_i is a sequence of tokens $x_{i,1}, x_{i,2}, ... x_{i,k}$ for block size $k \geq 1$. Thereafter, all else remains the same as unit-recurrence transformers. As we will see in our experiments, random variations in the block size during training can improve generalization during inference.

Block recurrence with accumulate A block recurrence transformer with accumulation operates on blocks of multiple tokens at each step as above, but accumulates the hidden state h_{i-1} as below. Specifically, instead of forgetting h_{i-1}, it is distributed forward into the working states.

$$h_1 = T(x_1) \tag{4}$$
$$(y_i, h_{i+1}) = T(0.5h_{i-1} + h_i, 0.5h_{i-1} + x_{i+1}) \tag{5}$$
$$y_N = T(h_N) \tag{6}$$

Optionally, any of the above recurrent transformer variants may have multiple layers.

Computational Complexity

Let L, D, N and K be the number of layers, the embedding dimension, the sequence length and the recurrence block size respectively. Table 1 shows the computational complexity in terms of floating point operations per sample.

Table1: Computational Complexity	
Architecture	**Flops/sample**
Regular transformer with quadratic attention	$L(N^2D + ND^2)$
Recurrent transformer with forget ($K = 1$)	$LN(4D + 2D^2)$
Block recurrent transformer with forget	$L(N/K)(4DK^2 + 2KD^2)$
Block recurrent transformer with accumulate	$L(N/K)(4DK^2 + 2KD^2 + 2)$

While recurrent transformers are inherently sequential, they are memory-efficient since they only need two blocks in working memory. Therefore, batch-sizes can be large during training and thereby amenable to parallel processing.

Experimental Results

All of our experiments were run on an M4 Mac Mini with 16GB memory. Unless stated otherwise, the learning rate was set at 5E-4 with the Adam optimizer. Code available at
https://github.com/bnkausik/recurrent_transformer

Unit recurrence with forget

We compare the performance of the recurrent transformer against regular transformers on the nanoGPT Shakespeare data set of Karpathy (2023). Table 2 specifies the base transformer layer in our comparison.

Table 2: Base Transformer Layer: LLM			
Context length	**Embedding Dimension**	**Number of heads**	**Dropout**
256	384	6	0.2

Table 3 shows the results of our experiments comparing the performance of a 1-layer and 6-layer regular transformer operating on the full sequence length, against a 1-layer recurrent transformer with and without positional embeddings added to the token embeddings. The validation loss entries for the recurrent transformer shows the best achieved and in parentheses, the validation loss closest to that achieved by the regular transformer along with the corresponding training cost. It is clear that the recurrent transformer outperforms the regular transformer on this dataset, with comparable training cost by vastly reduced inference cost.

Table 3: Experimental Results: LLM

Model	**Parameters**	**Batch size**	**Flops per sample**	**Best Val. Loss**	**Standard Error**	**Training flops**
Regular Transformer - 1 layer	1.89M	128	6.29E+07	1.5697	2.23E-04	1.75E10
Regular Transformer - 6 layers	10.74M	64	3.77E+08	1.4815	3.67E-04	1.3E10
Recurrent Transformer with positional embedding	1.89M	256	7.59E+07	1.4738 (1.481)	2.16E-04	2.4E10 (1.55E10)
Recurrent Transformer - no positional embedding	1.79M	256	7.59E+07	1.4699 (1.483)	2.23E-04	2.3eE10 (1.43E10)

Figs (2a) and (2b) show the validation loss and training loss for the four models in Table 2. The horizontal axis is the number of training flops, while the vertical axis is the logarithmic loss.

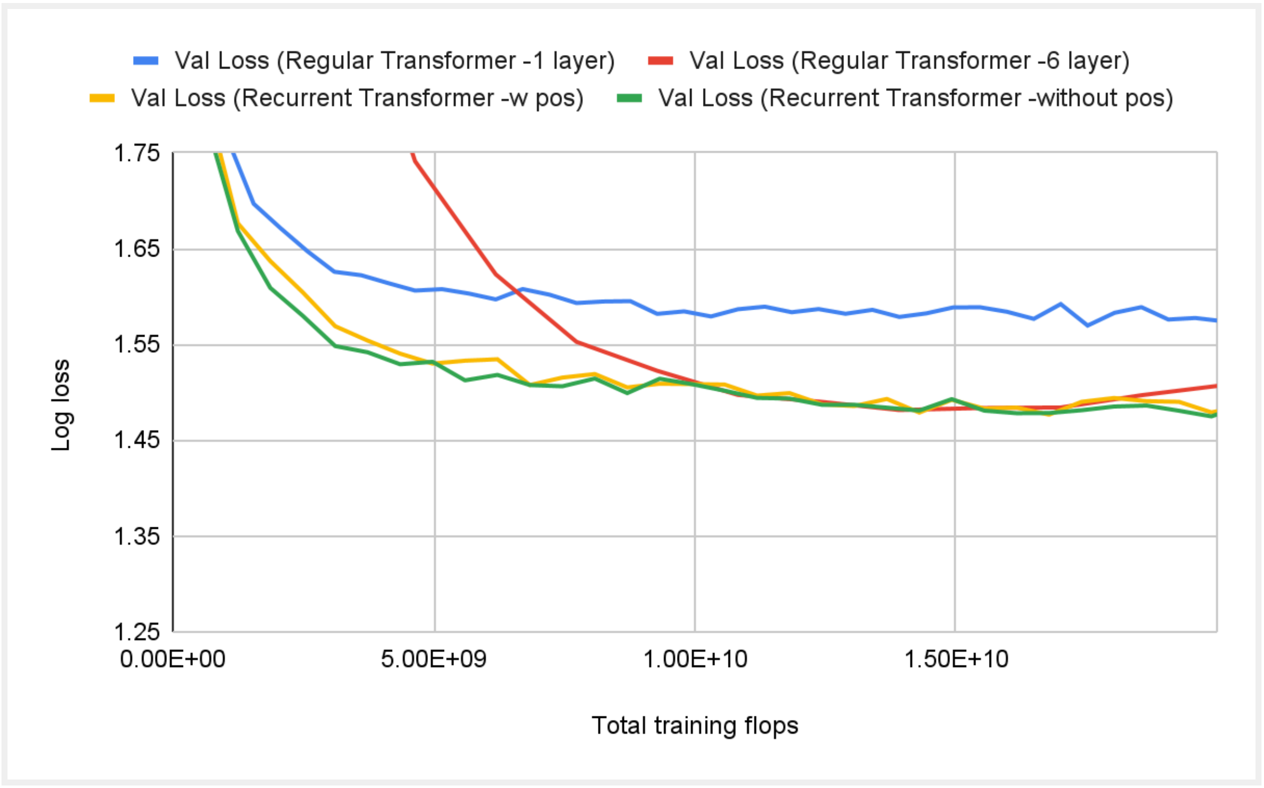


Fig. 2(a): Validation loss for the four models of Table 2.

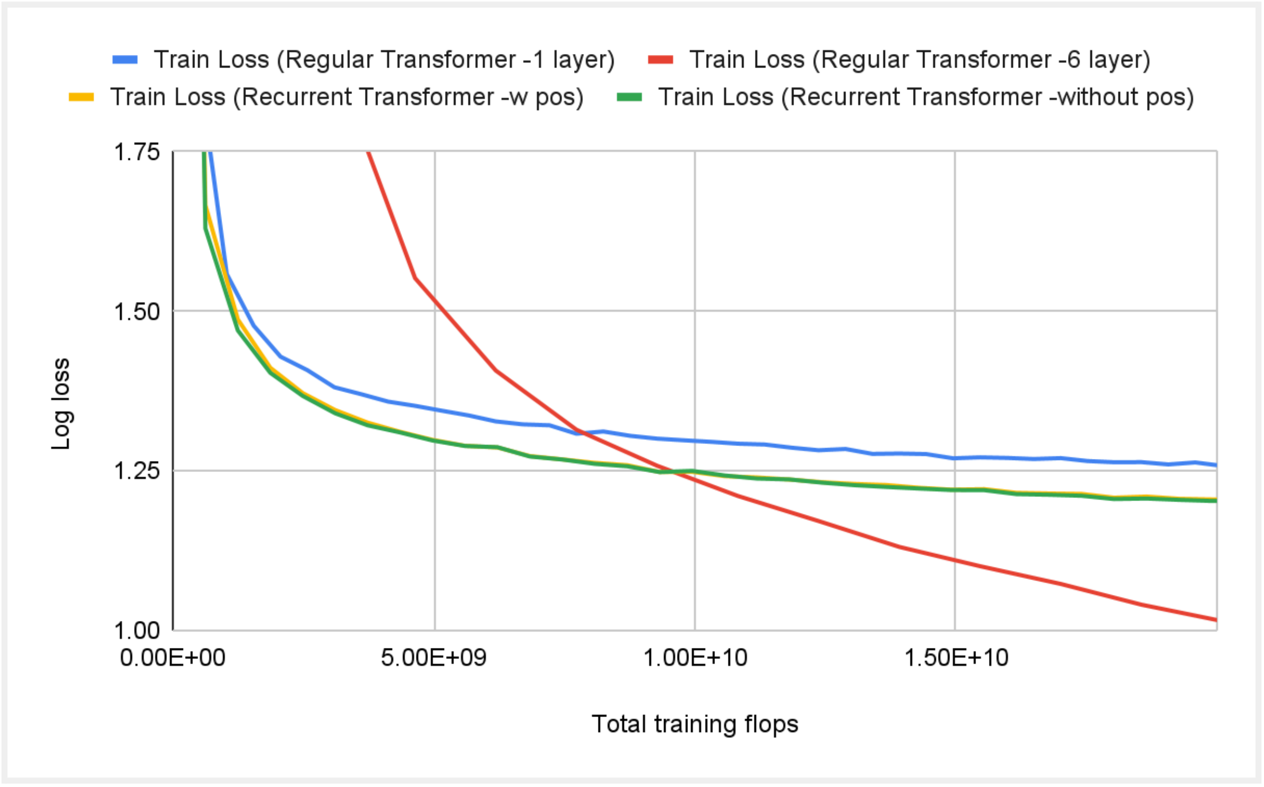


Fig. 2(b): Training loss for the four models of Table 2.

Block recurrence with forget

We compare the performance of the recurrent transformer against regular transformers on the "Long Range Arena" image classification dataset of Tay et al (2020):

> *"This task is an image classification task, where the inputs are sequences of pixels. In other words, an $N \times N$ image is flattened to a sequence of length N^2 pixels. ...To simplify the setup, we map the input images to a single gray-scale channel where each pixel is represented with an 8-bit pixel intensity (vocabulary size of 256). In LRA, we use the CIFAR-10 dataset..."*

Specifically, the CIFAR-10 dataset, Krishevsky & Hinton (2009), comprises a training set of 50,000 32x32 pixel images, and 10,000 test images across 10 categories. For this task, the images are converted into 256 grayscale values and fed to the transformer as an input sequence of 1024 tokens across a vocabulary of 256. The output is a single token across a vocabulary of 10 items representing the classification label of the images. Table 4 specifies the base transformer layer in our comparison.

Table 3: Base Transformer Layer: CIFAR-10

Context length	Embedding Dimension	Number of heads	Dropout
1024	32	4	0.05

Table 4: Experimental Results: CIFAR-10 grayscale classification

Model	Parameters	Batch size	Flops per sample	Best Val. Loss	Standard Error	Val. Accuracy	Training flops
Regular Transformer (4 layers)	90K	40	1.38E08	1.615	1.8e-02	42.9%	8.69E13
Block recurrent Transformer (16-48 random blocks)	53.6K	256	6.29E06	1.631	7.08E-03	43.09%	8.18E13

Table 4 shows the results of our experiments on this dataset. For the recurrent transformer, the block size was chosen uniformly randomly in the range 16 to 48 during training, but fixed at the mean value of 32 during validation. The random block sizes substantially improves the generalization capability of the recurrent transformer on this dataset, akin to the traditional dropout mechanism. The validation accuracy reported in Table 4 is the classification accuracy corresponding to the best validation loss.

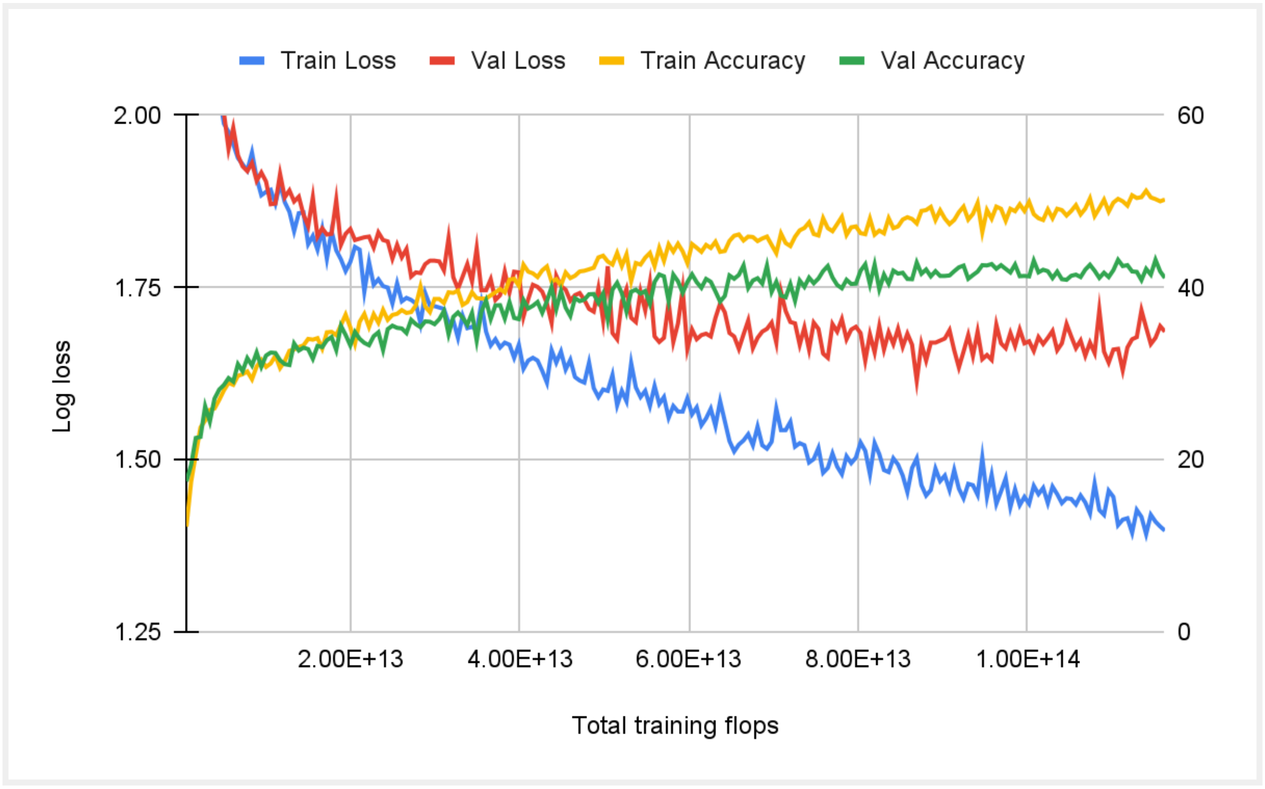


Fig. 3(a): Loss and accuracy for the regular transformer of Table 4.

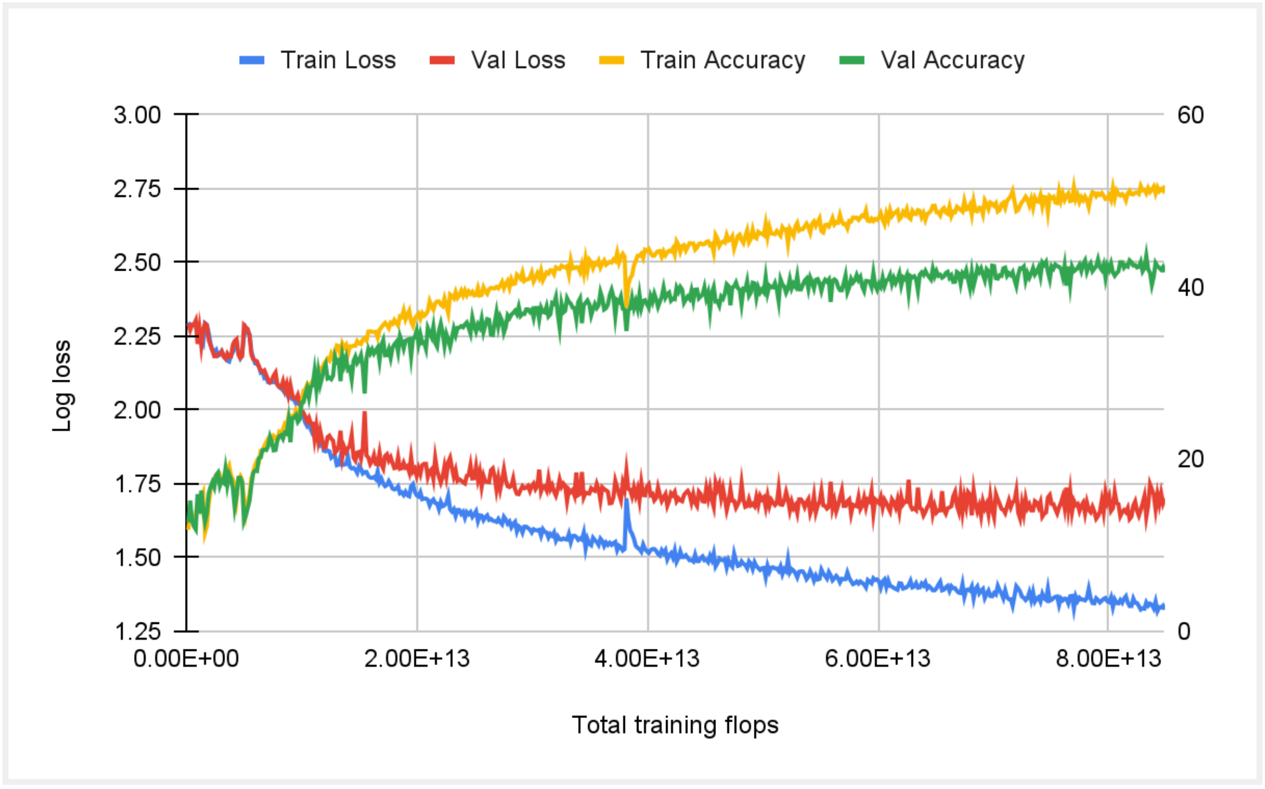


Fig. 3(b): Loss and accuracy for the random block recurrence transformer of Table 4.

Block recurrence with accumulate

We test the performance of the recurrent transformer on the copy and selective copy tasks of Gu & Dao (2024). The copy task involves 16 tokens, followed by 4096 noise tokens, followed by 16 tokens to trigger recall of the first 16 tokens, for a total sequence of 4028 tokens. The tokens are drawn from an alphabet of 16 unique characters, of which 2 characters are reserved for the noise and recall tokens respectively. Successful copy would reproduce the first 16 tokens in the original order against the last 16 recall tokens. The selective copy task randomly intersperses the 16 tokens to be copied amongst 4096 noise tokens, followed by 16 tokens to trigger recall of the first 16 tokens, for a total sequence of 4028 tokens.. Successful copy would reproduce the 16 interspersed tokens in their original order against the last 16 recall tokens.

Copy and selective copy tasks are challenging for recurrent networks as reported in the literature, e.g., Gu & Dao (2024), Ren et al (2025). In our experiments, the surprisingly small recurrent transformer of Table 5 solves the tasks.

Table 5: Block Recurrence Transformer with Accumulate (508.7K parameters)				
Context length	**Embedding Dimension**	**Number of heads**	**Block Size**	**Dropout**
4128	96	6	32	0.05

In training the above network, we found that progressively increasing the number of noise tokens allowed for rapid convergence even on a small model. Specifically, the training process begins with 32 noise tokens, trains until the validation loss drops down to say 0.3, then doubles the number of noise tokens to 64. This process is repeated until the number of noise tokens reaches 4096, at which point the training process drives the validation loss down to the desired minimal level for completion. The learning rate for the Adam optimizer is a function of the context length N and validation loss as below:

$$lr = 5E\text{-}4(1 - 0.5N/4128)$$
$$If\ N\ is\ 4218:$$
$$decayed\_lr = lr\left(1 - e^{-validation\ loss}\right)$$

Table 6 shows the results and Figs 4(a) and 4(b) show the loss and accuracy for the selective copy task during the training process. The spikes in the figures are where the number of noise tokens doubles during training. The flops/sample reported in Table 6 is when each sample has 4096 noise tokens, which is the case during inference.

Table 6: Experimental Results: Copy & Selective Copy								
Model	**Batch size**	**Num. Samples**	**Flops per Sample**	**Val. Loss**	**Std Err Val Loss**	**Val. Accuracy**	**Std. Err Val Acc**	**Training flops**
Copy Task	64	5.76E4	1.27E08	0.006	6.75E-05	100%	1.67E-03	1.32E13
Selective Copy Task	64	1.11E+06	1.27E08	0.022	8.29E-04	99.6%	2.45E-02	6.72E+15

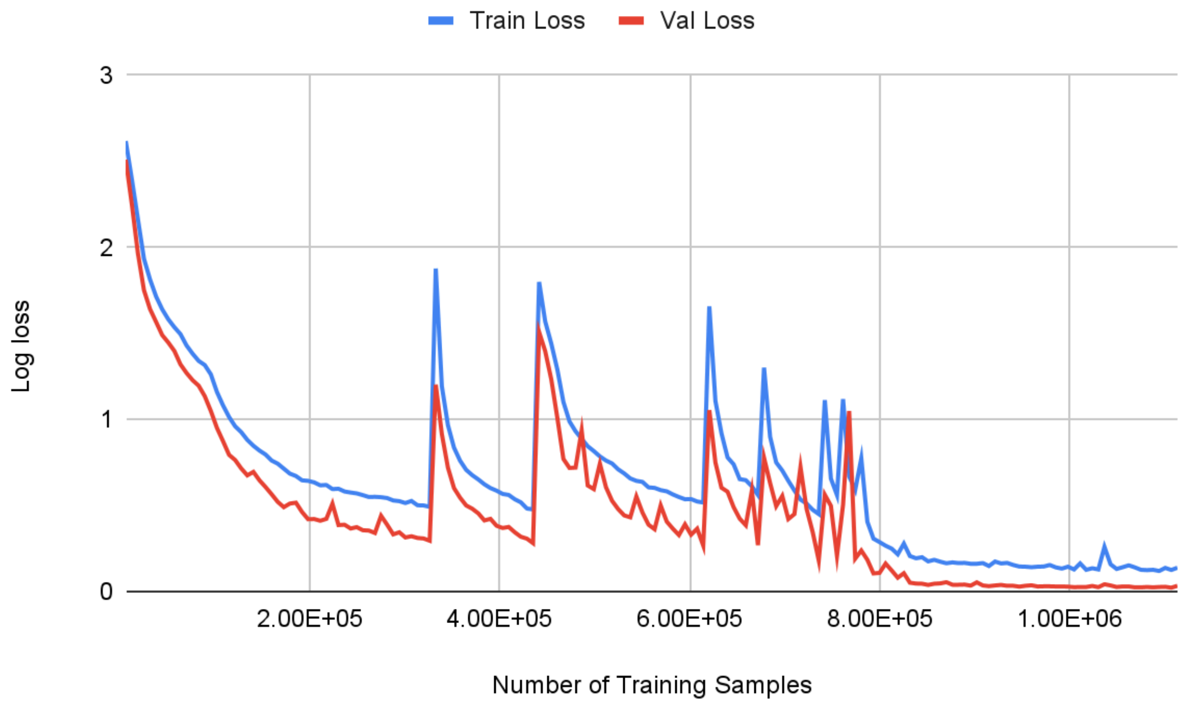


Fig. 4(a): Loss on the selective copy task for the block recurrent transformer with accumulate of Table 5.

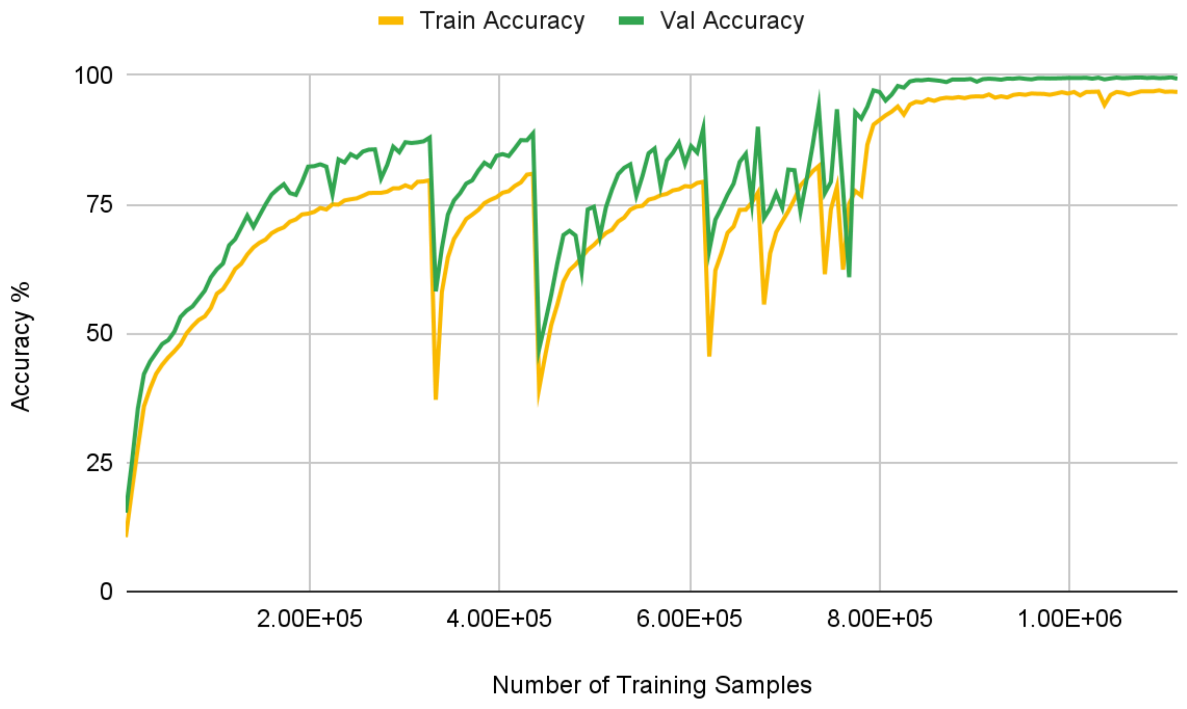


Fig. 4(b): Accuracy on the selective copy task for the block recurrent transformer with accumulate of Table 5.

Summary

We propose recurrent transformers, a neural network architecture that combines the simplicity and efficiency of recurrent networks with the efficacy of transformers. Specifically, a recurrent transformer is a transformer that is progressively applied to a fixed-width sliding window across the input sequence, thereby operating in linear time in the length of the sequence during both training and inference. We present three variants (1) **unit recurrence with forget**, which operate on unit-width segments of the input sequence, progressively forgetting information, and suitable for natural language processing; (2) **block recurrence with forget**, which operate on blocks of the input sequence, progressively forgetting information, and suitable for processing multi-dimensional datasets; and (3) **block recurrence with accumulate**, which operate on blocks of the input sequence, progressively accumulating information, suitable for copy and selective copy tasks. In our experiments across each of these variants, we find that a single layer recurrent transformer can meet or beat the performance of multi-layer transformers on benchmark datasets, delivering comparable accuracy at the same training cost, but at vastly reduced inference cost. One limitation of our work is the need for distinct forget and accumulate variants for different tasks. A second limitation is the need for validation on much larger natural language tasks.

References

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30
2. LeCun, Yann, John Denker, and Sara Solla. "Optimal brain damage." *Advances in neural information processing systems* 2 (1989).
3. Hassibi, B., Stork, D. G., & Wolff, G. J. (1993, March). Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks* (pp. 293-299). IEEE.
4. Han, S., Pool, J., Tran, J., & Dally, W. (2015). Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, *28*.
5. Liu, Z., Sun, M., Zhou, T., Huang, G., & Darrell, T. (2018). Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*.
6. Blalock, D., Gonzalez Ortiz, J. J., Frankle, J., & Guttag, J. (2020). What is the state of neural network pruning?. *Proceedings of machine learning and systems*, *2*, 129-146
7. Sun, M., Liu, Z., Bair, A., & Kolter, J. Z. (2023). A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*.
8. Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., & Peste, A. (2021). Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. Journal of Machine Learning Research, 22(241), 1-124.
9. Frantar, E., & Alistarh, D. (2023, July). Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning* (pp. 10323-10337). PMLR.
10. Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.
11. Chen, G., Choi, W., Yu, X., Han, T., & Chandraker, M. (2017). Learning efficient object detection models with knowledge distillation. *Advances in neural information processing systems*, *30*.
12. Asami, T., Masumura, R., Yamaguchi, Y., Masataki, H., & Aono, Y. (2017, March). Domain adaptation of DNN acoustic models using knowledge distillation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 5185-5189). IEEE.
13. Kausik, B. N. (2024). Occam Gradient Descent. arXiv preprint arXiv:2405.20194.
14. Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In International Conference on Machine Learning, pages 4055–4064. PMLR, 2018.
15. Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. arXiv preprint arXiv:1904.10509, 2019.
16. Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. arXiv preprint arXiv:2004.05150, 2020.
17. Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. arXiv preprint arXiv:2001.04451, 2020.
18. Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. Sparse sinkhorn attention. In International Conference on Machine Learning, pages 9438–9447. PMLR, 2020.
19. Irwan Bello, William Fedus, Xianzhi Du, Yann Dauphin, Ashish Srinivas, Tengyu Zhou, Zihang Barret, and Bryan Zoph. Lambdanetworks: Modeling long-range interactions without attention. In International Conference on Learning Representations, 2021.

20. Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Belanger, Lucy Sainath, et al. Rethinking attention with performers. arXiv preprint arXiv:2009.14794, 2021.
21. Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah Smith, and Lingpeng Kong. Random feature attention. arXiv preprint arXiv:2103.02114, 2021.
22. Zhe Zheng, Da-Cheng Juan, Yi Tay, Dara Bahri, and Donald Metzler. Randomized attention with linear complexity. In Advances in Neural Information Processing Systems, 2022.
23. Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. arXiv preprint arXiv:2006.04768, 2020.
24. Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nystr¨omformer: A nystr¨om-based algorithm for approximating self-attention. arXiv preprint arXiv:2102.03902, 2021.
25. Xiaoyu Ma, Yiding Wen, Yuanmeng He, Hai Zhao Liu, Dianhai Liu, Jianhua Li, and Xuan Dong. Nested attention for long document language modeling. arXiv preprint arXiv:2105.11875, 2021.
26. Ma, X., Zhou, C., Kong, X., He, J., Gui, L., Neubig, G., ... & Zettlemoyer, L. (2022). Mega: Moving average equipped gated attention. arXiv preprint arXiv:2209.10655.
27. Josh Alman and Zhao Song. Fast attention requires bounded entries. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023, 2023
28. Ma, X., Yang, X., Xiong, W., Chen, B., Yu, L., Zhang, H., ... & Zhou, C. (2024). Megalodon: Efficient llm pretraining and inference with unlimited context length. Advances in Neural Information Processing Systems, 37, 71831-71854.
29. Insu Han, Rajesh Jayaram, Amin Karbasi, Vahab Mirrokni, David P. Woodruff, and Amir Zandieh. Hyperattention: Long-context attention in near-linear time. In The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net, 2024
30. Kannan, R., Bhattacharyya, C., Kacham, P., & Woodruff, D. P. (2024). LevAttention: Time, Space, and Streaming Efficient Algorithm for Heavy Attentions. arXiv preprint arXiv:2410.05462.
31. Elman, J.L, (1990). Finding structure in time. Cognitive Science, 14(2):179–211, 1990. ISSN 0364-0213.
32. Hochreiter, S and Schmidhuber, J, (1997), Long short-term memory. Neural Computation, 9(8):1735–1780, 1997.
33. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.
34. Gu, A., Goel, K., & Ré, C. Efficiently modeling long sequences with structured state spaces. arXiv 2021. arXiv preprint arXiv:2111.00396.
35. Feng, L., Tung, F., Ahmed, M. O., Bengio, Y., & Hajimirsadeghi, H. (2024). Were RNNs all we needed?. arXiv preprint arXiv:2410.01201.
36. Karpathy, A., (2023), https://github.com/karpathy/nanoGPT/blob/master/README.md

37. Tay, Yi, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. "Long range arena: A benchmark for efficient transformers." arXiv preprint arXiv:2011.04006 (2020).
38. Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.
39. Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. arXiv preprint arXiv:2312.00752, 2024.
40. Ren, R., Li, Z., & Liu, Y. (2024). Exploring the Limitations of Mamba in COPY and CoT Reasoning. arXiv preprint arXiv:2410.03810.