RetFormers: Hybrid Attention-Retention Mechanisms for Faster Inference

Ishank Agrawal*
Massachusetts Institute of Technology
ishank@mit.edu

Alex Hu*
Massachusetts Institute of Technology
alexhu@mit.edu

Abstract

The transformer is the most successful language modeling architecture. The context length of these models needs to be expanded to support longer conversations. Yet the runtime and memory requirements during training and inference scale quadratically in the number of tokens modeled due to the self-attention module. This quadratic complexity is expensive compared to other architectures like RNNs, which only require linear runtime during inference. Reducing the latency of transformer inference is crucial for deploying these models in any real application, from consumer chatbots to automated agents. We address this challenge by combining the retentive module, which performs linear inference, with transformer self-attention modules, creating three variants. We find that inserting a small number of retentive layers after the transformer layers reaches the lowest perplexity with fixed compute. Larger-scale experiments are needed to confirm our findings.

1 Introduction

Transformers have been very successful at many language related tasks including language modeling and machine translation. Further, since attention can be computed through direct matrix multiplications: transformers are easier to train than previous RNN based architectures. However, transformers suffer from quadratic inference time complexity, which makes them hard to scale with sequence length.

There have been several architectures proposed to resolve this computational bottleneck with varying degrees of success. Models include RWKV (Peng et al., 2023), S4 (Gu et al., 2022), and RetNet (Sun et al., 2023). While these have sub quadratic inference time, their performance is still not close enough to transformers to be significant.

*equal contribution

Both RetNet and transformers use stacks of layers consisting of an attention sub-layer and a feed forward sub-layer. These similarities allow us to mix both attention heads and retention heads with much flexibility. In this paper we explore several combinations of retention and attention heads and how they affect computational complexity and performance on language modeling tasks.

2 Background

2.1 Queries, Keys, and Values

Both transformers and retentive networks use queries and key-value pairs to contextualize the sequence. Attention is any function that uses these three values to generate an output value is known, however in general we will use attention to refer to scaled dot-product attention proposed in (Vaswani et al., 2023)

Thus each attention/retention head assigns queries Q, keys K and values V to each token in the vocabulary. However they compute their outputs differently.

2.2 Attention

The scaled dot product attention output is given by

$$\text{ATTN}(Q, K, V) = \text{SOFTMAX}\left(\frac{QK^T}{\sqrt{d}}\right)V \quad (1)$$

h such heads are combined to create a multi-headed attention layer given by

$$\text{MHA}_h(X) = \text{CONCAT}(\text{ATTN}_1, \ldots \text{ATTN}_h) \quad (2)$$

here ATTN_i is the ith head, and each head has their own associated Q, K, V mappings.

2.3 Retention

The retentive attention output is given by

$$\text{RETN}_{\theta,\gamma}(Q,K,V) = \left[\left((Q \odot \Theta)(K \odot \bar{\Theta})^T\right) \odot D\right] V \tag{3}$$

$$\Theta_n = e^{in\theta}, \quad D_{nm} = \begin{cases} \gamma^{n-m} & \text{if } n \geq m \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

h such heads are combined to create a multi-scaled retention layer, which is given by

$$Y = \text{GN}(\text{CONCAT}(\text{RETN}_1(X), \ldots \text{RETN}_h(X)) \tag{5}$$

$$\text{MSR}_h(X) = \text{SWISH}(X) \odot Y \tag{6}$$

where GN, SWISH refer to GroupNorm (Wu and He, 2018) and Swish (Ramachandran et al., 2017) layers respectively.

We now explore various hybrid models that use these attention and retention heads in conjunction, hoping to get a model with performance similar to that of transformers but with faster inference time.

3 Models

We propose three different types of hybrid retention-attention transformer, Retformer, architectures. Each of these vary in the respective placement of the attention and retention heads. For each of these types of models, we have an associated attention-retention ratio or AR-ratio. This is defined as the ratio of the number of attention heads to retention heads in the entire architecture and is a good proxy to estimate the relative multiply-accumulate (MAC) operations required during inference.

Each of our reformers has an overall architecture similar to the original transformer in (Vaswani et al., 2023). We use L layers, h heads in each layer and a hidden dimension size of d_{model}. We also include rotation positional embeddings (RoPE) (Su et al., 2023) to include positional information into the network.

3.1 Type-A retformer

In a type-A retformer, the first few layers are multi-headed attention layers, while the remaining last layers are multi-scaled retention layers. Formally, for a type-A retformer withAR-ratio χ, let $L_A = \frac{\chi}{1+\chi} L$ layers. Then the ith layer is given by

$$Y^i = \begin{cases} \text{LN}(\text{MHA}(X^i) + X^i) & \text{if } i \leq L_A \\ \text{LN}(\text{MSR}(X^i) + X^i) & \text{otherwise} \end{cases} \tag{7}$$

$$X^{i+1} = \text{LN}(\text{FNN}(Y^i) + Y^i \tag{8}$$

Here LN stands for LayerNorm, and FFN corresponds to the feedforward networks used. (Ba et al., 2016), and is used similarly to the original transformer paper.

3.2 Type-B retformer

Our type-B retformer is designed nearly identically, however the first $L_R = \frac{1}{1+\chi} L$ layers are now multi-scaled retention layers and the rest are multi-headed attention. More formally, we have

$$Y^i = \begin{cases} \text{LN}(\text{MSR}(X^i) + X^i) & \text{if } i \leq L_R \\ \text{LN}(\text{MHA}(X^i) + X^i) & \text{otherwise} \end{cases} \tag{9}$$

$$X^{i+1} = \text{LN}(\text{FNN}(Y^i) + Y^i \tag{10}$$

3.3 Type-C Retformer

In this case, we merge attention and retention heads within each layer. Thus for a given AR-ratio χ, let the number of attention and retention heads be $h_A = \frac{\chi}{1+\chi} h$ and $h_R = \frac{1}{1+\chi} h$ respectively. Then we define a hybrid reformer layer (HRL) as

$$\text{HRL}(X) = \text{CONCAT}(\text{MHA}_{h_A}(X), \text{MSR}_{h_R}(X)) \tag{11}$$

Similar to before, our entire network is then given by

$$Y^i = \text{LN}(\text{HRL}(X^i) + X^i) \tag{12}$$

$$X^{i+1} = \text{LN}(\text{FNN}(Y^i) + Y^i) \tag{13}$$

4 Experiments

4.1 Training Data

We evaluate on the WIKITEXT-103 (Merity et al., 2016) benchmark. We use the GPT-2 (Radford et al., 2019) tokenizer for our input representation. Since our corpus does not contain any document structure, we partitioned our corpus into contiguous blocks of 128 tokens for both the training and the evaluation texts. This is similar to the approach taken in (Baevski and Auli, 2019).

4.2 Models

We trained 3 different models, due to compute limitations, differing in their arrangement of attention and retention heads. We trained a vanilla transformer, retnet, and a mixed transformer that contains one retention layer at the end following the rest attention layers. For each of our models we used $L = 12$ layers with $h = 12$ heads and a hidden size $d_{\text{model}} = 768$. We used a maximum sequence length of 128 tokens which is the same as the length of our input sequences.

4.3 Hyperparameter Configuration

We use the AdamW optimizer (Loshchilov and Hutter, 2017) with $\beta_1 = 0.9, \beta_2 = 0.98$ and weight decay 0.05. We warm-started the networks over the first five epochs to a learning-rate of 10^{-3} and then cool-downed the learning rate to 10^{-4} afterwards. We train each model for at least 40 epochs with distributed data parallelism with total batch size 144. All our models had roughly 150 million parameters, including the embedding modules.

5 Results

5.1 Main results

We observe the following train and validation loss curves.

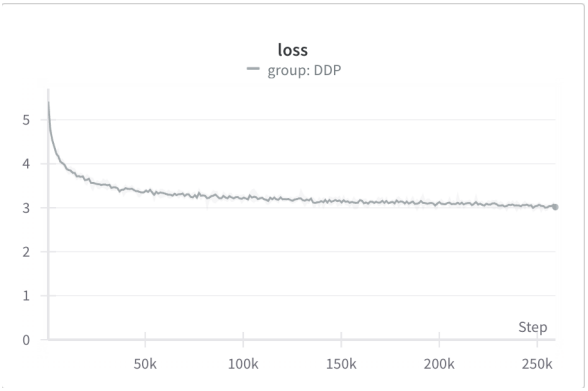

Figure 1: Retnet train loss vs training step

5.2 Additional results

Moreover, we train our model on a down-sized version of the dataset with 2048 chunks of length 128, batch size 32, 8 heads per layer, and learning rate $1e-3$ with Adam across all possible architectures for 10 epochs. We used such a small dataset due to compute constraints. We evaluate our models based on train perplexity since our validation perplexity is not stable as we do not have enough training

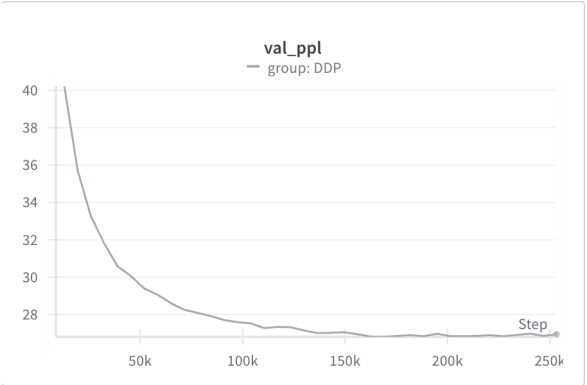

Figure 2: Retnet per-epoch validation perplexity

Figure 3: Transformer train loss vs training step

samples. We observe that type A is superior to type B which is superior to type C. These experiments suggest that replacing the final layer by a retention layer could be useful, and give rough heuristics for which architectures could perform best at a larger scale.

In addition, we evaluate our model by profiling the number of MACs (Multiply-ACCumulate) operations as the model runs in total during inference on a training batch and dividing by the batch size. We observe a roughly 50% increase in MACs when going RetNet to Transformer, and this change is smooth as we increase as expected. For type-A and type-B retformers, the trend is linear in the percentage of attention layers, and for type-C, the trend seems to be super-linear, probably due to the extra operations we added in type-C retformers to project the transformer and retnet head outputs together. We do not assume the use of a KV-Cache in our experiments. If we do use a KV-Cache, the parts of the model spent in attention will become more efficient, so the improvement in inference latency for more Retnet-leaning architectures versus more transformer-leaning architecture will increase.

Architecture Type	**AR ratio**	**# of GigaMACs**	**Train perpl.**
Vanilla Transformer	1:0	10.305	4.251
Vanilla RetNet	0:1	6.709	23.984
Type-A Retfomer	1:11	7.009	13.426
Type-A Retfomer	2:10	7.308	34.811
Type-A Retfomer	3:9	7.608	6.942
Type-A Retfomer	4:8	7.908	5.162
Type-A Retfomer	5:7	8.207	5.885
Type-A Retfomer	6:6	8.507	4.845
Type-A Retfomer	7:5	8.806	4.364
Type-A Retfomer	8:4	9.10	4.291
Type-A Retfomer	9:3	9.406	3.955
Type-A Retfomer	11:2	9.705	3.209
Type-A Retfomer	11:1	10.005	4.762
Type-B Retfomer	1:11	7.004	12.725
Type-B Retfomer	2:10	7.304	9.483
Type-B Retfomer	3:9	7.604	7.508
Type-B Retfomer	4:8	7.904	11.826
Type-B Retfomer	5:7	8.204	10.549
Type-B Retfomer	6:6	8.504	8.345
Type-B Retfomer	7:5	8.804	4.81
Type-B Retfomer	8:4	9.104	4.74
Type-B Retfomer	9:3	9.405	5.31
Type-B Retfomer	10:2	9.705	5.226
Type-B Retfomer	11:1	10.005	4.807
Type-C Retfomer	1:7	8.563	40.775
Type-C Retfomer	2:6	8.732	13002.334
Type-C Retfomer	3:5	9.012	119.673
Type-C Retfomer	4:4	9.406	206.809
Type-C Retfomer	5:3	9.911	72.801
Type-C Retfomer	6:2	10.529	52.354
Type-C Retfomer	7:1	11.260	56.122

Table 1: Training and test perplexities of various architectures

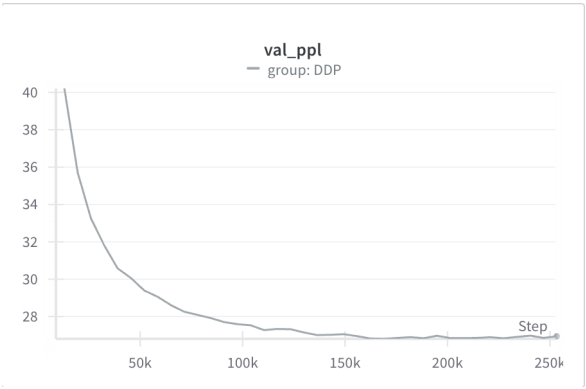

Figure 4: Transformer per-epoch validation perplexity

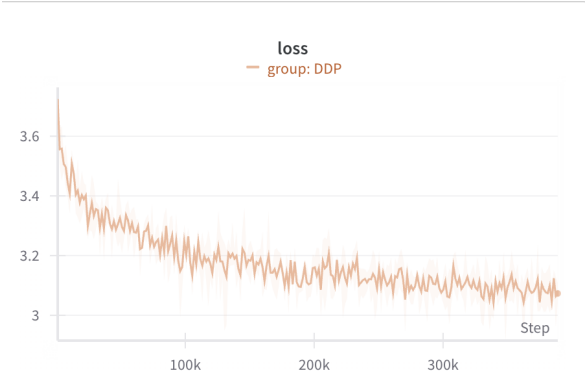

Figure 5: Mixed transformer train loss vs training step

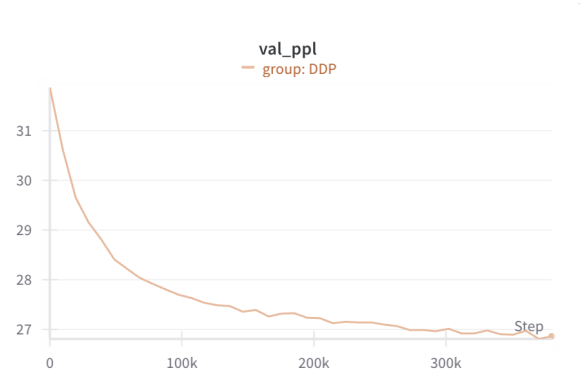

Figure 6: Mixed transformer per-epoch validation perplexity

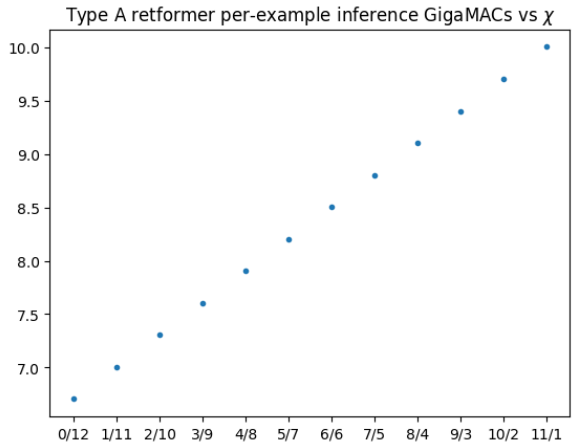

Figure 7

6 Conclusion

Our mixed transformer obtains a minimum validation perplexity of 26.808, our transformer obtains a minimum validation perplexity of 26.899, and retnet obtains a validation perplexity of 26.859. Surprisingly, our transformer does worse than the retnet. We suspect this is due to the lack of positional encodings in our training. On our smaller-scale experiments, we also find that type-A retformers with high attention-retention ratio reach the lowest perplexity, better than transformers.

We find these results promising and with more compute, the scaling of such retformers can be more thoroughly investigated.

7 Social and Ethical Concerns

Our research shows that similar perplexity levels can be achieved with less computation. Thus this research allows development of more efficient models, allowing people with less computational resources to experiment with larger language models.

Since our research works on LLM fundamentals, it faces similar ethical complications as general large language models. With good alignment and public policy, these ethical concerns can be mitigated.

Acknowledgements

This paper is the result of project work done in Prof. Yoon Kim's class 6.861 Quantitative Methods in Natural Language Processing. We are grateful for project Prof. Kim's guidance in this project as well as the support of the entire 6.861 course staff.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization.

Alexei Baevski and Michael Auli. 2019. Adaptive input representations for neural language modeling. In *International Conference on Learning Representations*.

Albert Gu, Karan Goel, and Christopher Ré. 2022. Efficiently modeling long sequences with structured state spaces.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization.

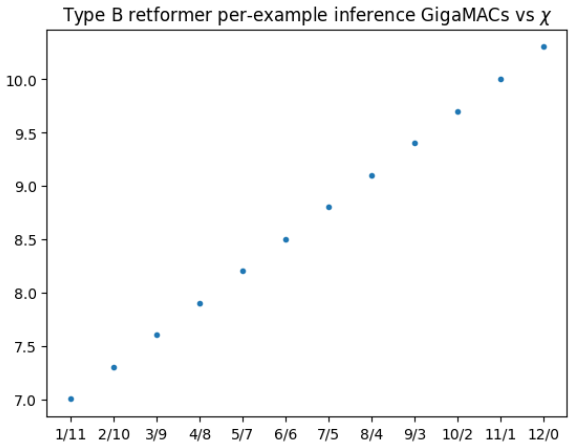

Figure 8

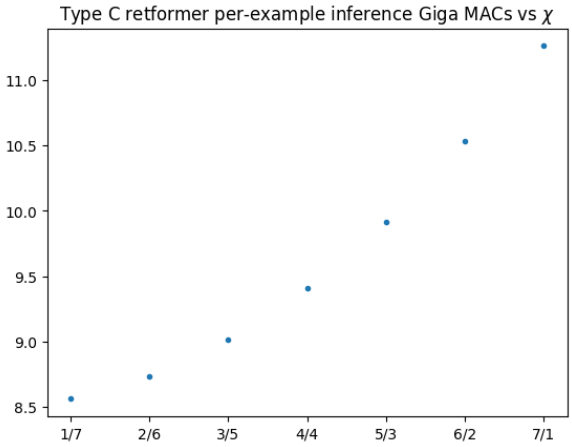

Figure 9

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartlomiej Koptyra, Hayden Lau, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Xiangru Tang, Bolun Wang, Johan S. Wind, Stansilaw Wozniak, Ruichong Zhang, Zhenyuan Zhang, Qihang Zhao, Peng Zhou, Jian Zhu, and Rui-Jie Zhu. 2023. Rwkv: Reinventing rnns for the transformer era.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. 2017. Searching for activation functions.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2023. Roformer: Enhanced transformer with rotary position embedding.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. 2023. Retentive network: A successor to transformer for large language models.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need.

Yuxin Wu and Kaiming He. 2018. Group normalization.