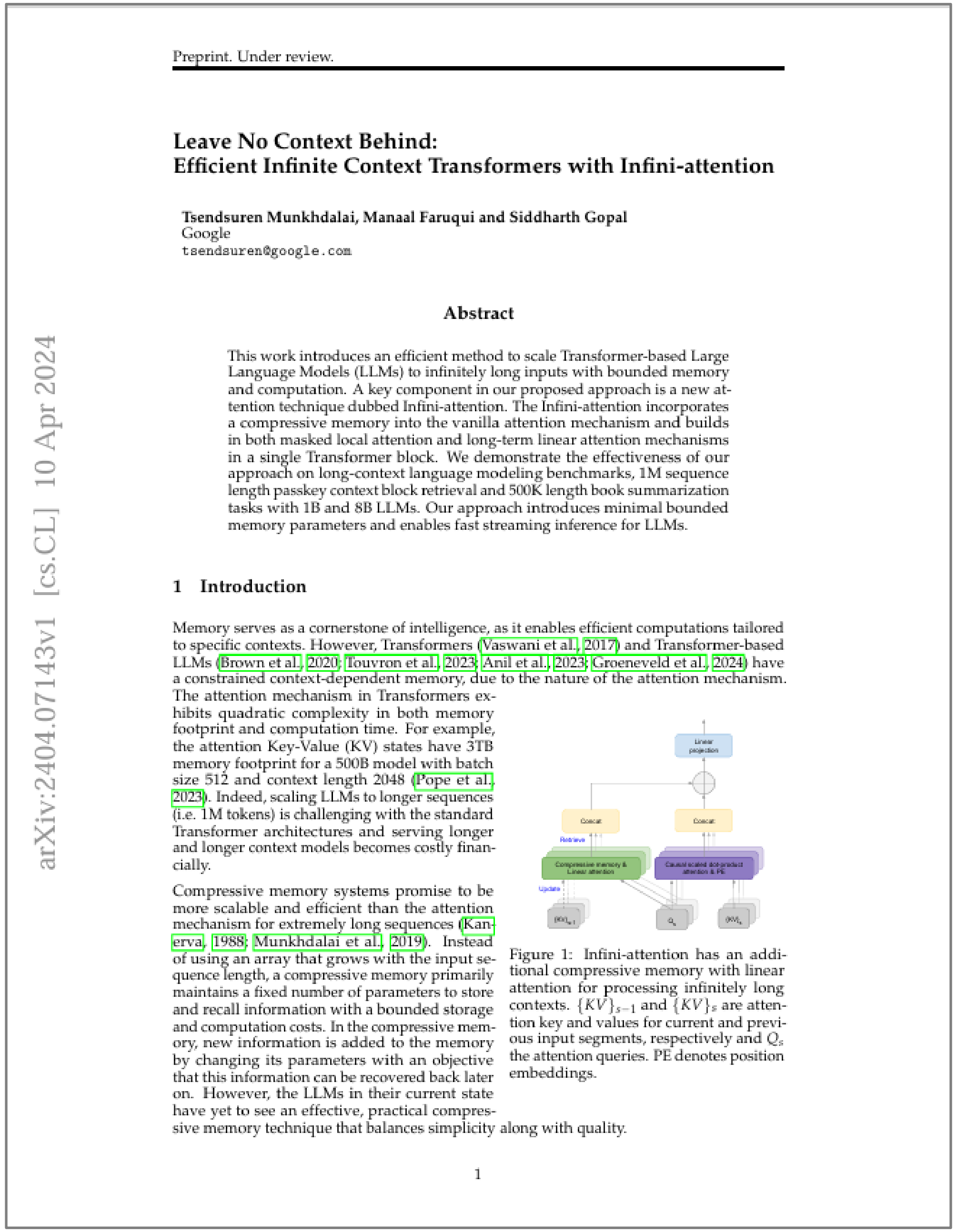
Leave No Context Behind: Efficient Infinite Context Transformers with Infini-attention

June 6, 2024

Leave No Context Behind: Efficient Infinite Context Transformers with Infini-attention

Tsendsuren Munkhdalai et al., Google

https://arxiv.org/abs/2404.07143



Leave No Context Behind: Efficient Infinite Context Transformers with Infini-attention

Tsendsuren Munkhdalai, Manaal Faruqui and Siddharth Gopal
Google
tsendsuren@google.com

Abstract

This work introduces an efficient method to scale Transformer-based Large Language Models (LLMs) to infinitely long inputs with bounded memory and computation. A key component in our proposed approach is a new attention technique dubbed Infini-attention. The Infini-attention incorporates a compressive memory into the vanilla attention mechanism and builds in both masked local attention and long-term linear attention mechanisms in a single Transformer block. We demonstrate the effectiveness of our approach on long-context language modeling benchmarks, 1M sequence length passkey context block retrieval and 500K length book summarization tasks with 1B and 8B LLMs. Our approach introduces minimal bounded memory parameters and enables fast streaming inference for LLMs.

1 Introduction

Memory serves as a cornerstone of intelligence, as it enables efficient computations tailored to specific contexts. However, Transformers (Vaswani et al., 2017) and Transformer-based LLMs (Brown et al., 2020; Touvron et al., 2023; Anil et al., 2023; Groeneveld et al., 2024) have a constrained context-dependent memory, due to the nature of the attention mechanism. The attention mechanism in Transformers exhibits quadratic complexity in both memory footprint and computation time. For example, the attention Key-Value (KV) states have 3TB memory footprint for a 500B model with batch size 512 and context length 2048 (Pope et al., 2023). Indeed, scaling LLMs to longer sequences (i.e. 1M tokens) is challenging with the standard Transformer architectures and serving longer and longer context models becomes costly financially.

Compressive memory systems promise to be more scalable and efficient than the attention mechanism for extremely long sequences (Kanerva, 1988; Munkhdalai et al., 2019). Instead of using an array that grows with the input sequence length, a compressive memory primarily maintains a fixed number of parameters to store and recall information with a bounded storage and computation costs. In the compressive memory, new information is added to the memory by changing its parameters with an objective that this information can be recovered back later on. However, the LLMs in their current state have yet to see an effective, practical compressive memory technique that balances simplicity along with quality.

Linear projection
Concat
Concat
Retrieve
Compressive memory & Linear attention
Causal scaled dot-product attention & PE
Update
{KV}s-1
Qs
{KV}s

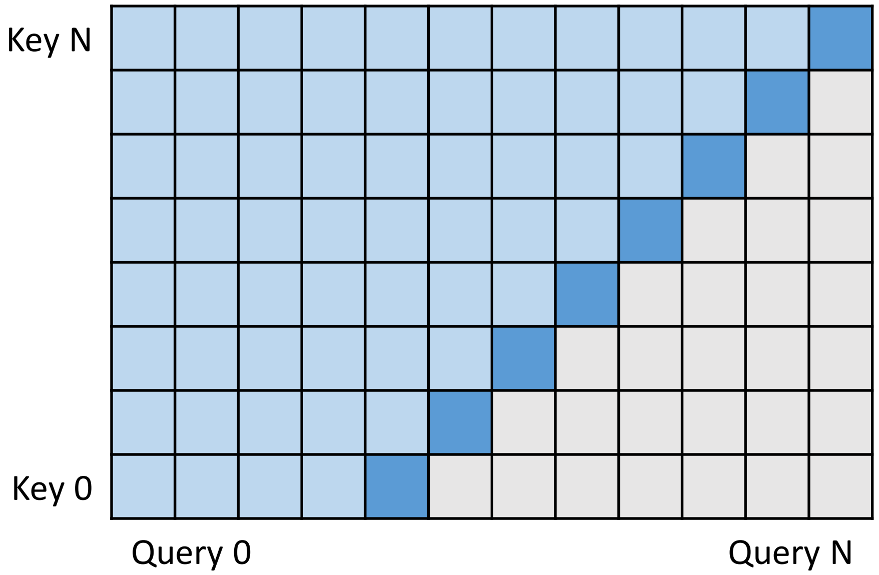
Figure 1: Infini-attention has an additional compressive memory with linear attention for processing infinitely long contexts. $\{KV\}_{s-1}$ and $\{KV\}_s$ are attention key and values for current and previous input segments, respectively and Q_s the attention queries. PE denotes position embeddings.

Infini-attention overview

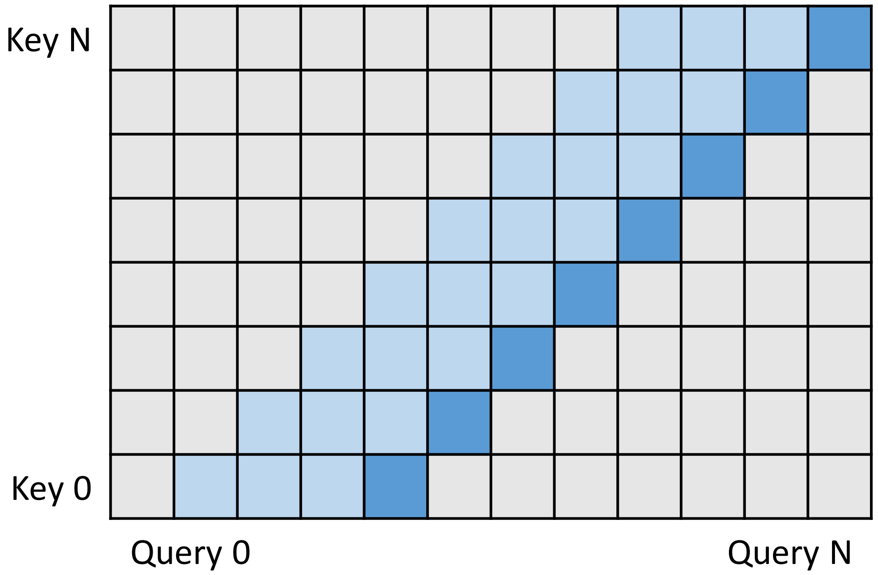
- Addresses long context scaling with transformers. Used by Gemini 1.5?
- Seems to combine 3 concepts:
  - Transformer XL divided text into segments and processed two consecutive segments at a time
  - Fast Weight Memory (coauthored by Munkhdalai) creates a compressive, associative memory using matrix multiplication
    - This associative memory uses keys to store values
    - Here, queries are used during retrieval, implicitly performing a similarity function
  - Linear Attention approximated attention with a recurrent calculation that afforded linear scaling
- The result is memory unit parameter count that is constant and processing cost that is linear with respect to the input length

Transformer-XL [1]

- Standard attention has quadratic cost for longer sequence length
- Windowed attention caps cost but can only attend to recent tokens



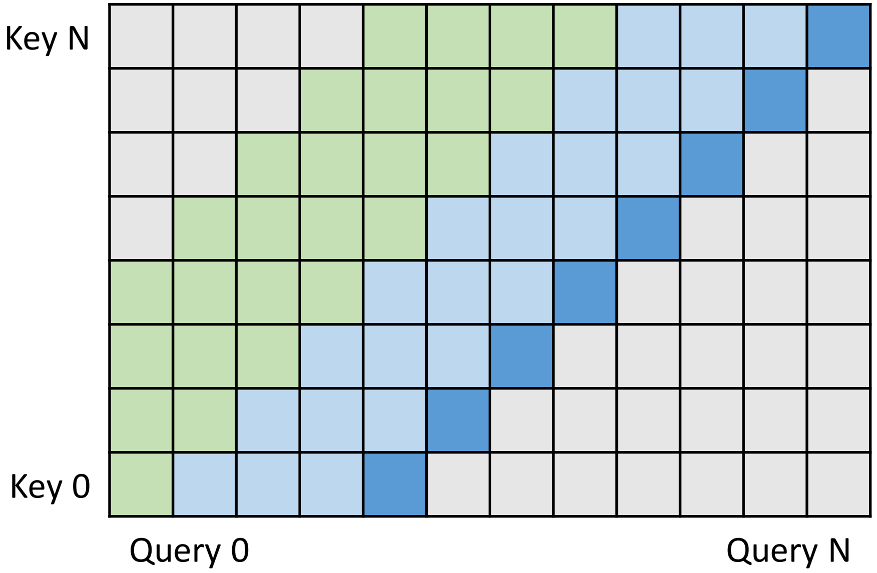
Causal masking for standard full attention



Causal masking for window (win=4) attention

Transformer-XL [2]

- Transformer-XL uses two segments of size *s*
- Within the current segment, normal attention is used
- Cached previous segment serves role of passing history forward



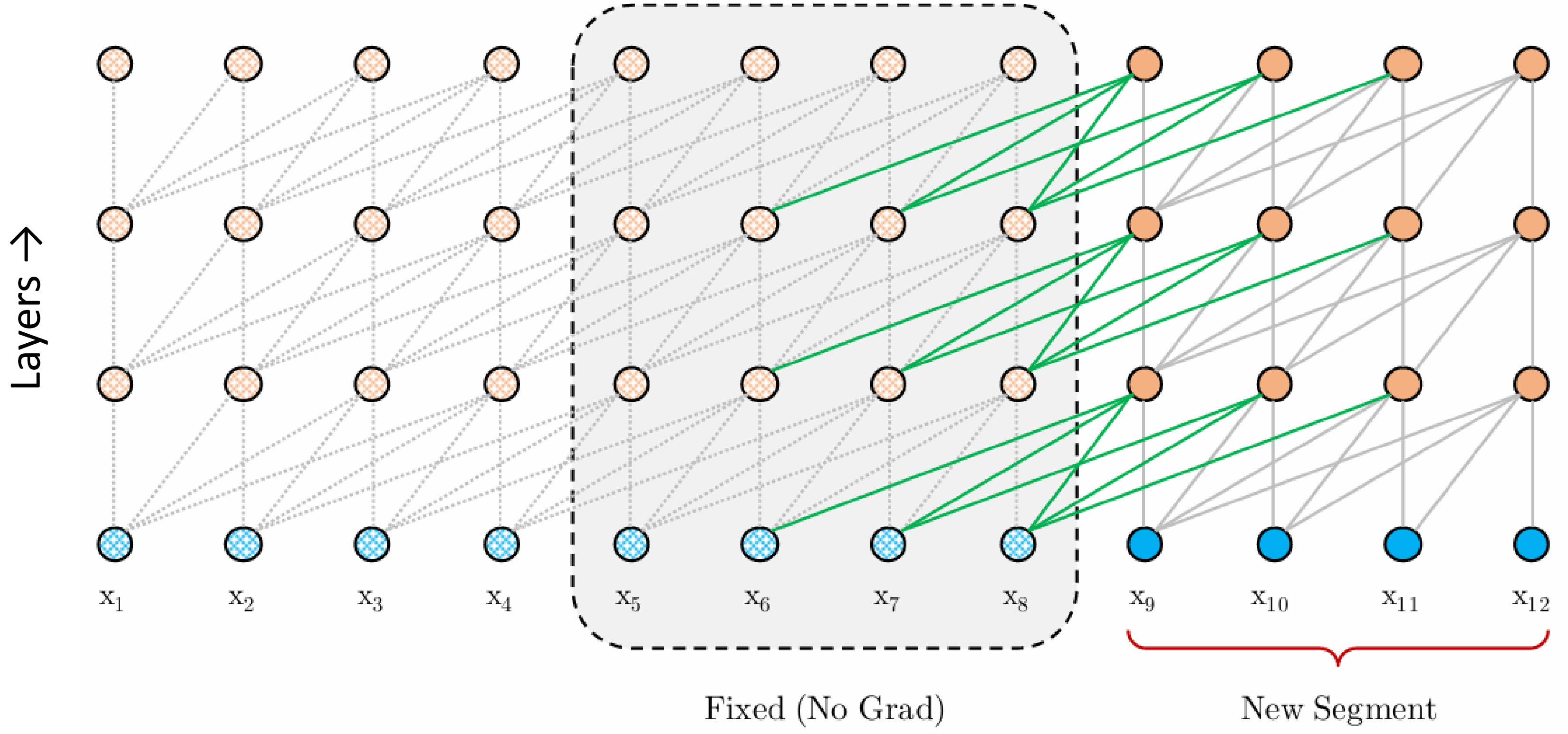
Causal masking with segments (s=4)

Transformer-XL [3]

- Queries are calculated with size *s* weight matrix, like windowed
- Cached keys and values from previous segment are concatenated with current ones, meaning they have double the parameters in their weight matrices
- The desire is for the weights for previous segment to function slightly differently than the regular attention weights
- There are a few more technicalities involving relative position embeddings instead of absolute position embeddings

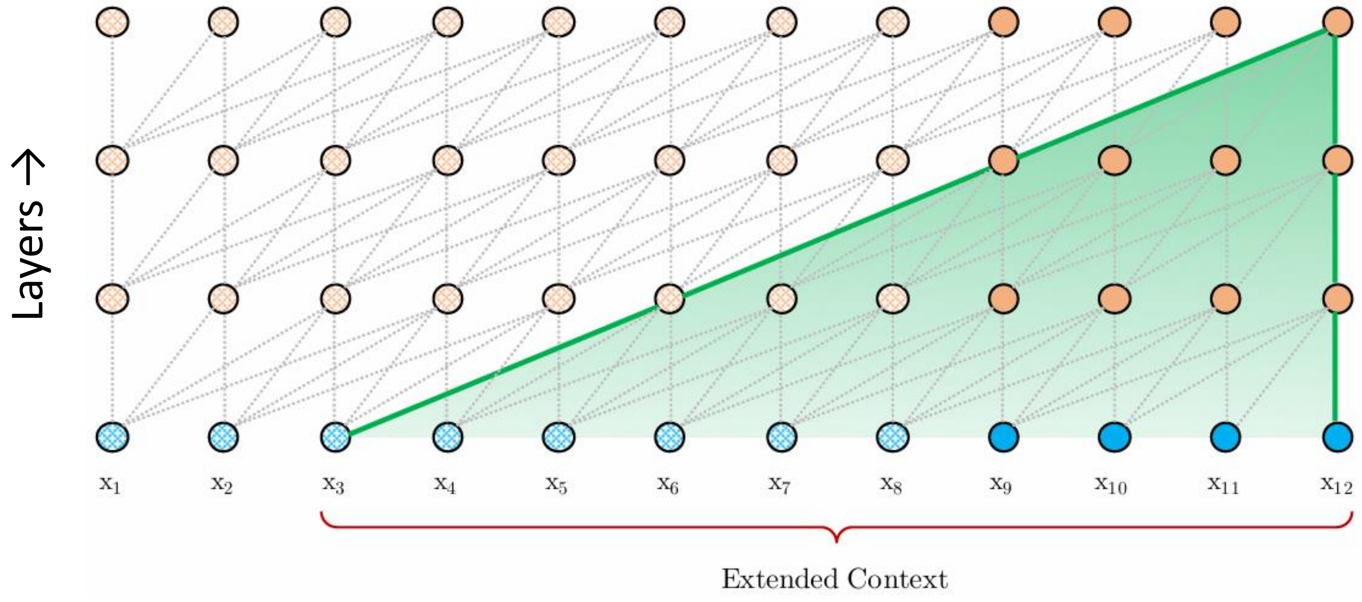
Transformer-XL [4]

- During training, note how the green connections in the diagram pass extra context information to the current segment



Transformer-XL [5]

- When we look at layer 1, the information token 12 gets from token 5 from layer 0 had information from tokens 1-4 baked into it
- As we go up layers, the effective context grows by 4 every layer up, like the receptive field in a CNN increases after each layer



Infini-attention method [1]

- Transformer-XL processed two actual input segments at a time, caching the previous segment and loading the current segment
- The key difference here is that we replace the previous segment with a linearized attention on keys stored in our memory
- Instead of propagating token information through layers the way Transformer-XL works, a memory can (in theory) immediately recall any key that it has ever seen

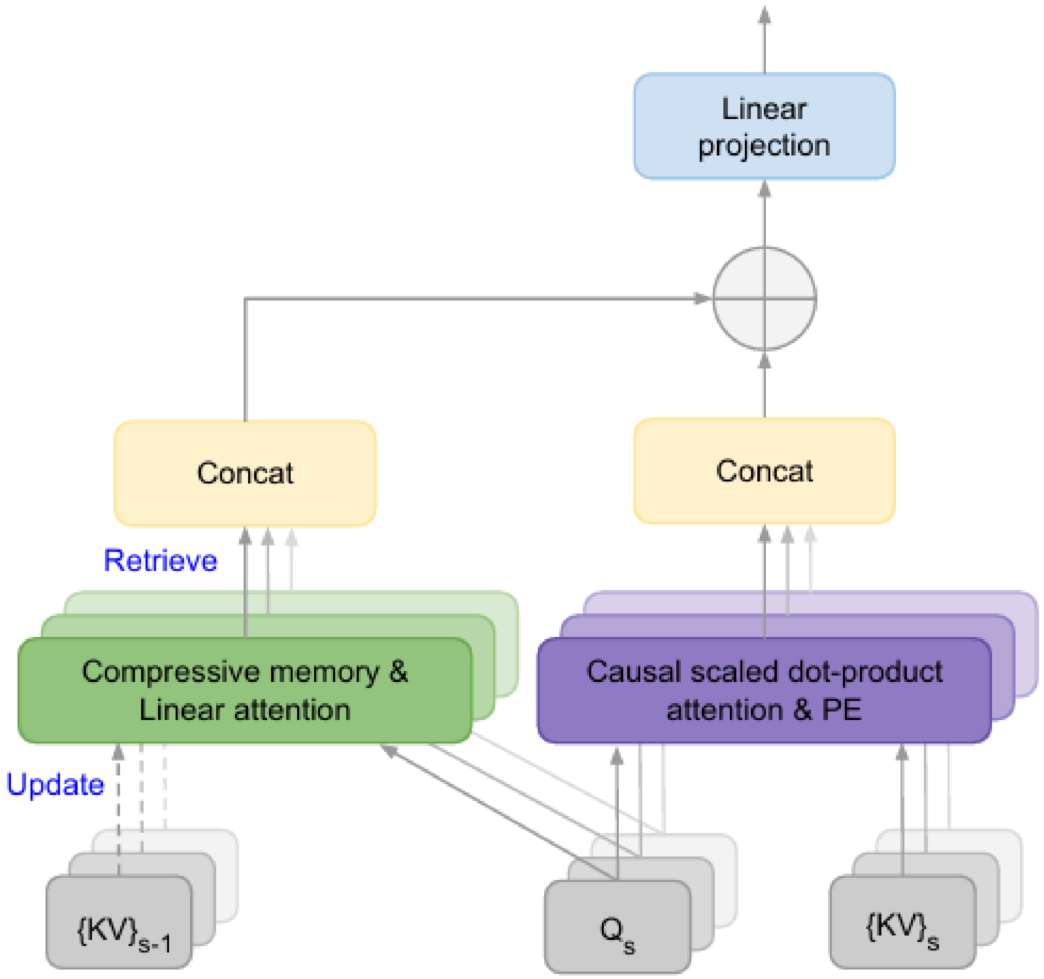
Memory and linear attention

- Start with a matrix M with dimensions d_{key} x d_{value} that is all zeros
- If you have row vectors k for your key and v for your value,
  then outer product $k^{\text{T}}v$ is a rank one matrix the same shape as M
- If you multiply your query q times M, $qM = q(k^{\text{T}}v) = (qk^{\text{T}})v$
  - This is like the dot product of our query and key times the value, just like attention
  - Note: if the query and key don't match, the dot product will be close to zero
- So, to do a write to memory, you add $k^{\text{T}}v$ to M
  To read, you multiply qM to get the values whose key(s) were most similar to the query
- Extra details in the paper include a $\sigma()$ to normalize q and k, doing all token positions at the same time using matrices instead of vectors, and tweaking the memory write to check to see if the value is already there (delta method)

Infini-attention method [2]

- Transformer-XL processed two actual input segments at a time, caching the previous segment and loading the current segment
- We replace the previous segment with the linearized attention using our memory mechanism, which is inexpensive
- Processing follows this new loop:
  - Run regular QKV attention on the current segment
  - Retrieve old memory contents using Queries, performing linear attention
  - The above two results are weighted summed with a learned parameter β
  - The result is output and also performs a memory unit update using Keys to store Values
  - The cumulative, updated memory is ready to be used with the next segment

Combining local attention with memory


Linear projection
Concat
Concat
Retrieve
Compressive memory & Linear attention
Causal scaled dot-product attention & PE
Update
{KV}$_{s-1}$
Q$_s$
{KV}$_s$

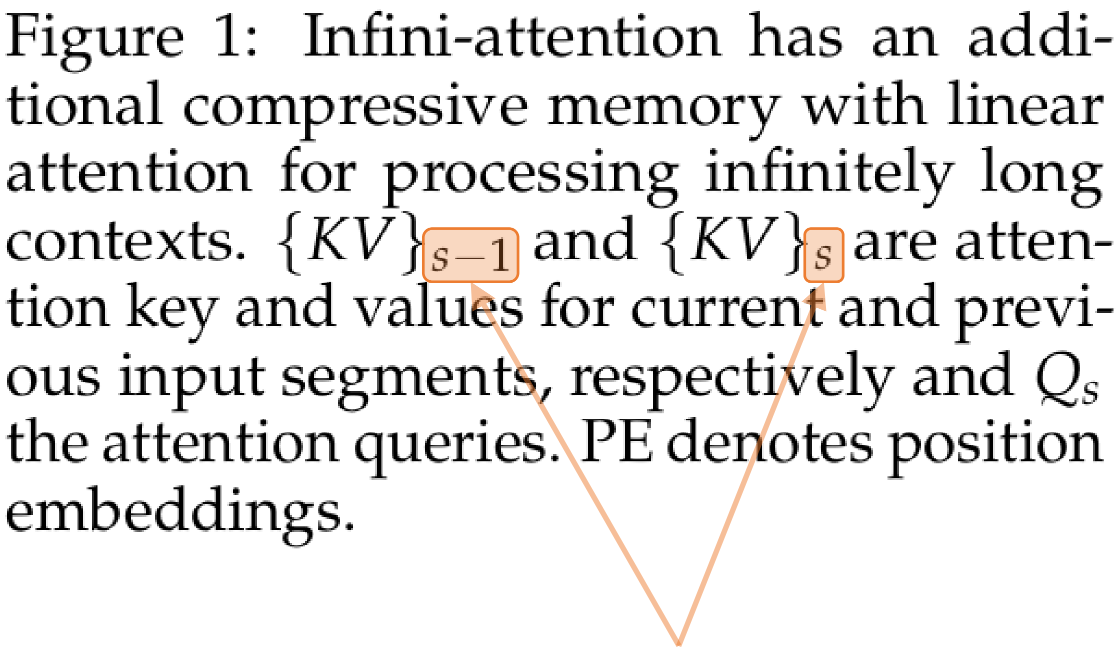


Figure 1: Infini-attention has an additional compressive memory with linear attention for processing infinitely long contexts. $\{KV\}_{\boxed{s-1}}$ and $\{KV\}_{\boxed{s}}$ are attention key and values for current and previous input segments, respectively and Q_s the attention queries. PE denotes position embeddings.

Note: I think they got s and s-1 backwards here

Infini-attention method [3]

- Transformer-XL processed two actual input segments at a time, caching the previous segment and loading the current segment
- We replace the previous segment with our memory mechanism
- There are two key improvements:
    - Our memory directly stores values, so it acts a lot more like attention than Transformer-XL propagating via the residual stream up through layers
    - Every layer, even the first layer, has access back all the way to the first token (see diagram on next slide)
- If you pick a reasonably big segment size, attention can work normally when you want to attend to a recent token (most of the time)
- If your memory works well (maybe this is a big if?), you can also attend to tokens that are far, far away in the context

Comparison of visible contexts

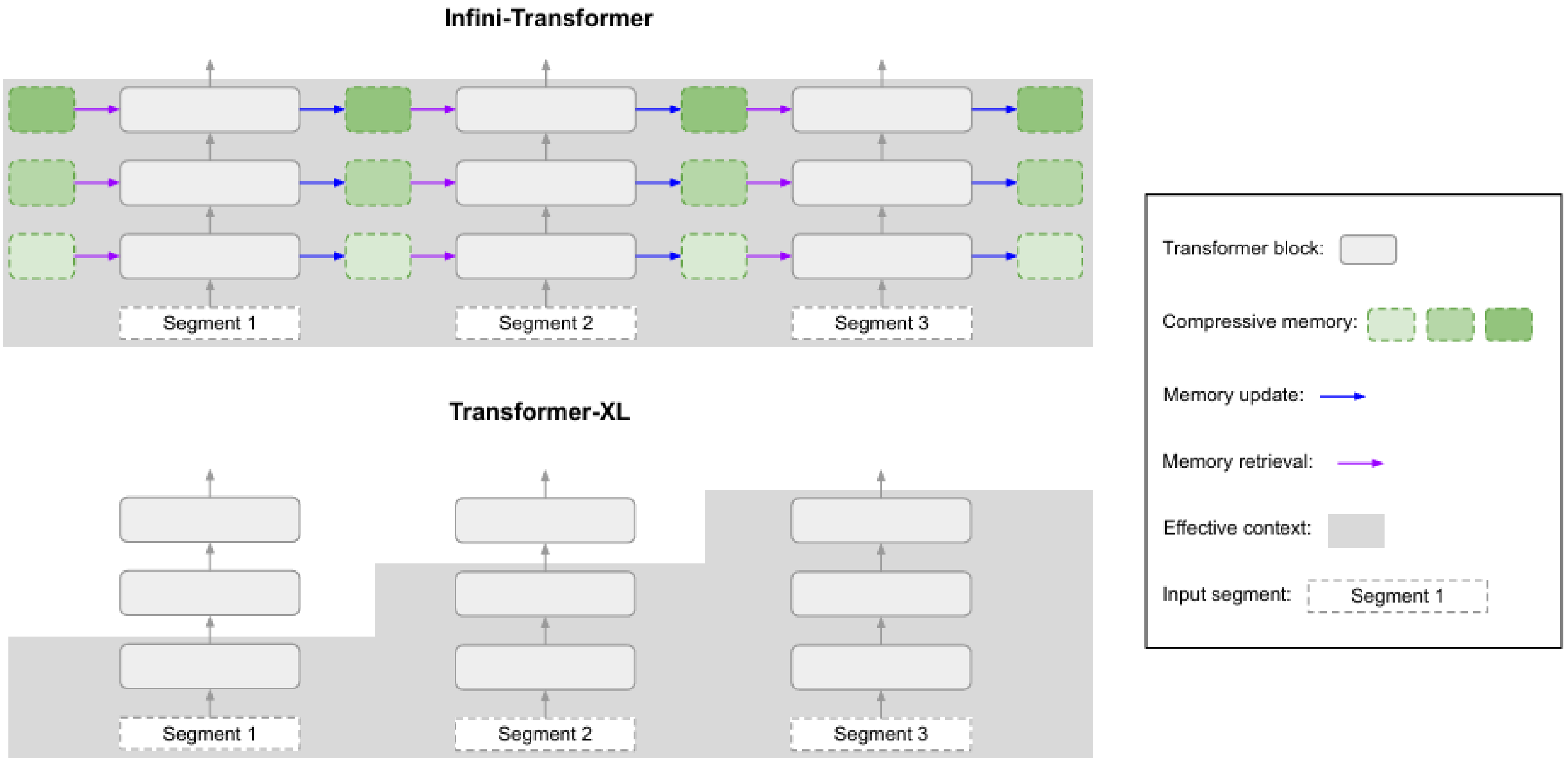
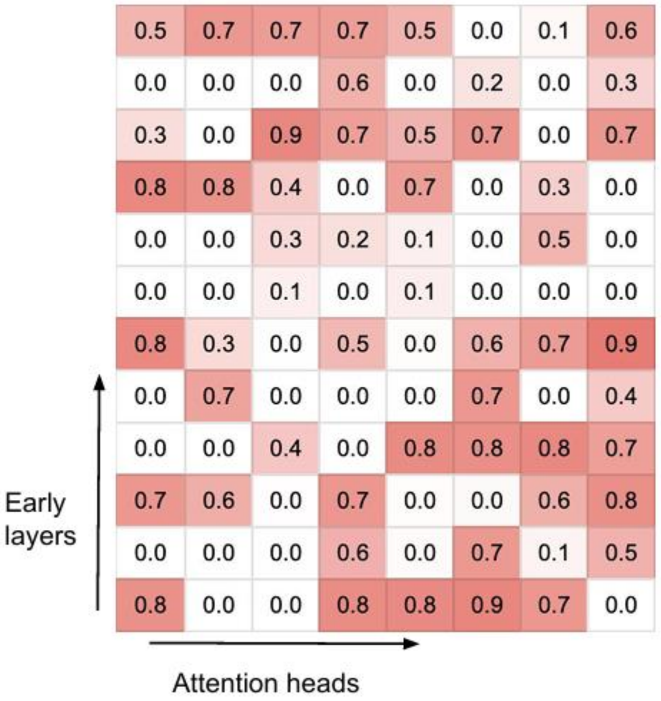


Figure 2: Infini-Transformer (top) has an entire context history whereas Transformer-XL (bottom) discards old contexts since it caches the KV states for the last segment only.

Attention heads focus differently

- Authors state that many attention head specialize in only local attention or memory attention, but other act as mixer heads

Values of sigmoid(β), where the β parameter is trained separately for each head of each layer

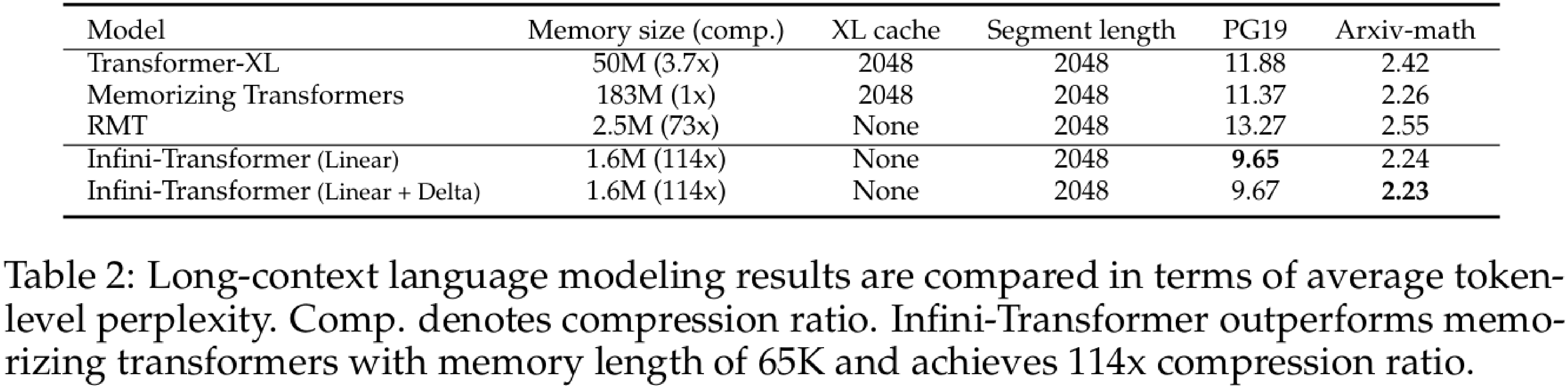


0.5	0.7	0.7	0.7	0.5	0.0	0.1	0.6
0.0	0.0	0.0	0.6	0.0	0.2	0.0	0.3
0.3	0.0	0.9	0.7	0.5	0.7	0.0	0.7
0.8	0.8	0.4	0.0	0.7	0.0	0.3	0.0
0.0	0.0	0.3	0.2	0.1	0.0	0.5	0.0
0.0	0.0	0.1	0.0	0.1	0.0	0.0	0.0
0.8	0.3	0.0	0.5	0.0	0.6	0.7	0.9
0.0	0.7	0.0	0.0	0.0	0.7	0.0	0.4
0.0	0.0	0.4	0.0	0.8	0.8	0.8	0.7
0.7	0.6	0.0	0.7	0.0	0.0	0.6	0.8
0.0	0.0	0.0	0.6	0.0	0.7	0.1	0.5
0.8	0.0	0.0	0.8	0.8	0.9	0.7	0.0

Figure 3: There are two types of heads emerged in Infini-attention after training: specialized heads with gating score near 0 or 1 and mixer heads with score close to 0.5. The specialized heads either process contextual information via the local attention mechanism or retrieve from the compressive memory whereas the mixer heads aggregate both current contextual information and long-term memory content together into single output.

Infini-attention results [1]

- They had better perplexity than some other models on long texts (not terribly compelling result)

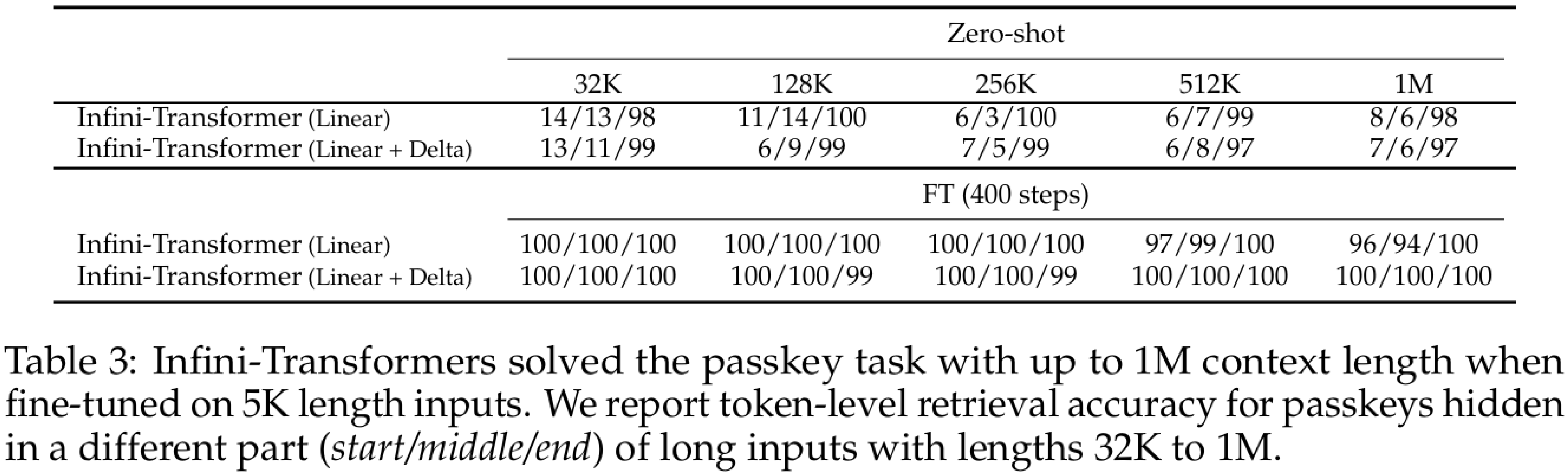


Model	Memory size (comp.)	XL cache	Segment length	PG19	Arxiv-math
Transformer-XL	50M (3.7x)	2048	2048	11.88	2.42
Memorizing Transformers	183M (1x)	2048	2048	11.37	2.26
RMT	2.5M (73x)	None	2048	13.27	2.55
Infini-Transformer (Linear)	1.6M (114x)	None	2048	**9.65**	2.24
Infini-Transformer (Linear + Delta)	1.6M (114x)	None	2048	9.67	**2.23**

Table 2: Long-context language modeling results are compared in terms of average token-level perplexity. Comp. denotes compression ratio. Infini-Transformer outperforms memorizing transformers with memory length of 65K and achieves 114x compression ratio.

Infini-attention results [2]

- They were able to solve the passkey task when fine-tuned on 5K length examples (passkey is a more interesting test)

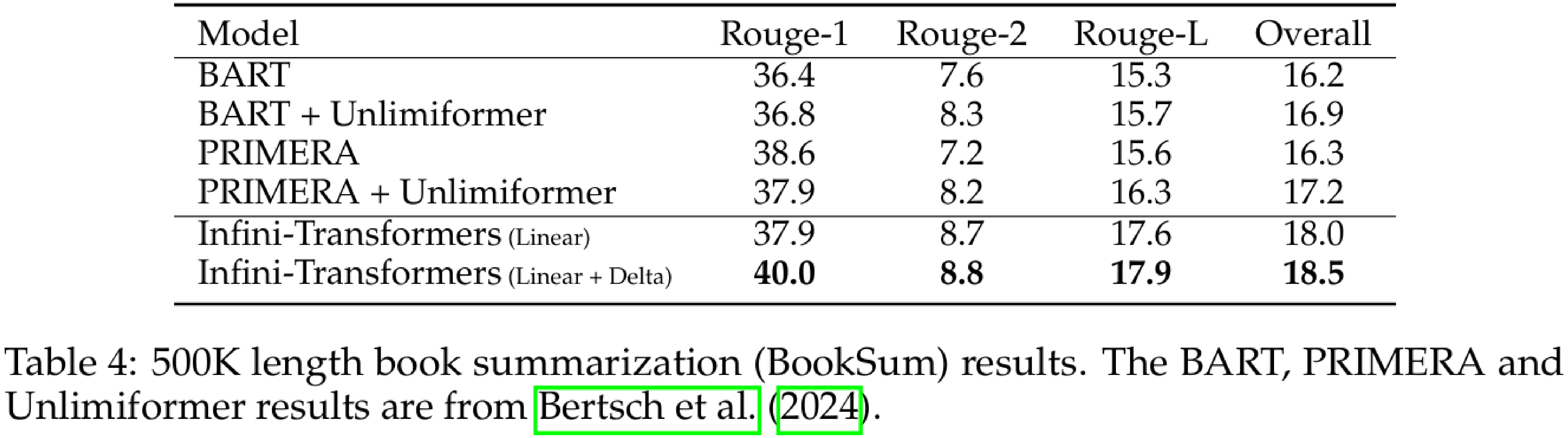


	Zero-shot				
	32K	128K	256K	512K	1M
Infini-Transformer (Linear)	14/13/98	11/14/100	6/3/100	6/7/99	8/6/98
Infini-Transformer (Linear + Delta)	13/11/99	6/9/99	7/5/99	6/8/97	7/6/97
	FT (400 steps)				
Infini-Transformer (Linear)	100/100/100	100/100/100	100/100/100	97/99/100	96/94/100
Infini-Transformer (Linear + Delta)	100/100/100	100/100/99	100/100/99	100/100/100	100/100/100

Table 3: Infini-Transformers solved the passkey task with up to 1M context length when fine-tuned on 5K length inputs. We report token-level retrieval accuracy for passkeys hidden in a different part (*start/middle/end*) of long inputs with lengths 32K to 1M.

Infini-attention results [3]

- They beat some encoder models on 500K length book summarization (not sure how hard this task is, and Rouge isn't a great metric)



Model	Rouge-1	Rouge-2	Rouge-L	Overall
BART	36.4	7.6	15.3	16.2
BART + Unlimiformer	36.8	8.3	15.7	16.9
PRIMERA	38.6	7.2	15.6	16.3
PRIMERA + Unlimiformer	37.9	8.2	16.3	17.2
Infini-Transformers (Linear)	37.9	8.7	17.6	18.0
Infini-Transformers (Linear + Delta)	**40.0**	**8.8**	**17.9**	**18.5**

Table 4: 500K length book summarization (BookSum) results. The BART, PRIMERA and Unlimiformer results are from Bertsch et al. (2024).

Infini-attention results [4]

- They show that their Rouge scores go up as the input length increases, demonstrating that they really are taking advantage of more text

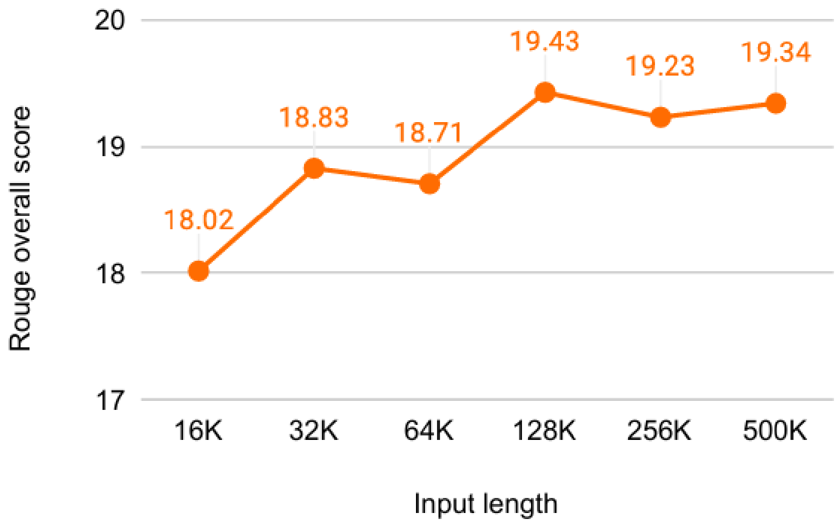


Figure 4: Infini-Transformers obtain better Rouge overall scores with more book text provided as input.

Infini-attention conclusion

- Transformer-XL had a simple scheme to break text into segments and process two at a time, but it didn’t perform that well
  - Information from 1st segment would propagate to the Lth segment in layer L
- Linear Attention Transformer had a nice linear scaling, but didn’t perform that well
- Infini-attention packs way more information into the previous segment than Transformer-XL
  - By using a memory, information from 1st segment is immediately available to all later segments in the same layer (to the extent that the memory and linear attention can work as well as full attention)
- The associative memory and linear attention have good scaling properties
  - Memory is constant, regardless of sequence length
  - Compute is linear with respect to sequence length

References [1]

- Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context
  Zihang Dai et al. (2019)
  https://arxiv.org/abs/1901.02860
- Learning Associative Inference Using Fast Weight Memory
  Imanol Schlag et al. (2020)
  https://arxiv.org/abs/2011.07831
- Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention
  Angelos Katharopoulos et al. (2020)
  https://arxiv.org/abs/2006.16236

References [2]

- TransformerFAM: Feedback attention is working memory
  Dongseong Hwang et al. (2024)
  https://arxiv.org/abs/2404.09173