Carnegie Mellon University

# Scaling Transformer to 1M tokens and beyond with RMT

Vedant Bhasin, Pin Qian, Prince Wang, Xinyu Yang

## Topic list

Carnegie Mellon University

## Topic list

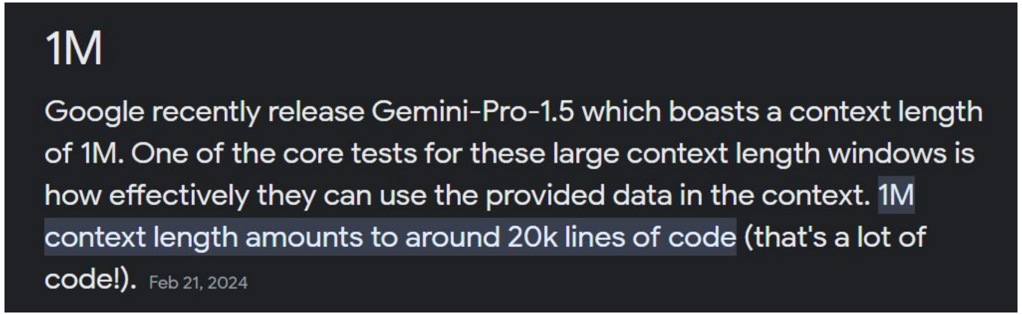
Carnegie Mellon University

## Why Long-Context LLMs?

One of the primary limitations of transformers is their ability to operate on long sequences of tokens.

For many applications of LLMs, overcoming this limitation is powerful.

- In Retrieval Augmented Generation (RAG), a longer context augments our model with more information.
- For Long-term planning tasks, such as chatbots, longer context means more capabilities.
- ……



1M

Google recently release Gemini-Pro-1.5 which boasts a context length of 1M. One of the core tests for these large context length windows is how effectively they can use the provided data in the context. 1M context length amounts to around 20k lines of code (that's a lot of code!). Feb 21, 2024

## Challenges in Achieving Long-Context Transformer-based LLMs

1. Attention Complexity
   - The length of an input sequence is limited by **quadratic time & memory complexity** of attention.
2. Max-Length Constraint
   - During training, determining the **max-length** is necessary, which is commonly set based on the available computational resources.
   - During inference, we also need to restrict the input length, since current Language Models exhibit noticeable performance degradation when handling input sequences exceeding max-length.

Carnegie Mellon University

## Challenges in Achieving Long-Context Transformer-based LLMs

3. In-Context Memory

- LLMs lack an memory mechanism. Global and local information has to be stored in the same **element-wise** representations. They rely on the **KV cache** to store representations of all previous tokens.
- Though this offers computational advantages in terms of parallelism, it presents challenges in tasks like chatbot applications, where **long-term memory retention** is essential.

## Topic list

Carnegie Mellon University

## Overview of Methods for Long-Context LLMs

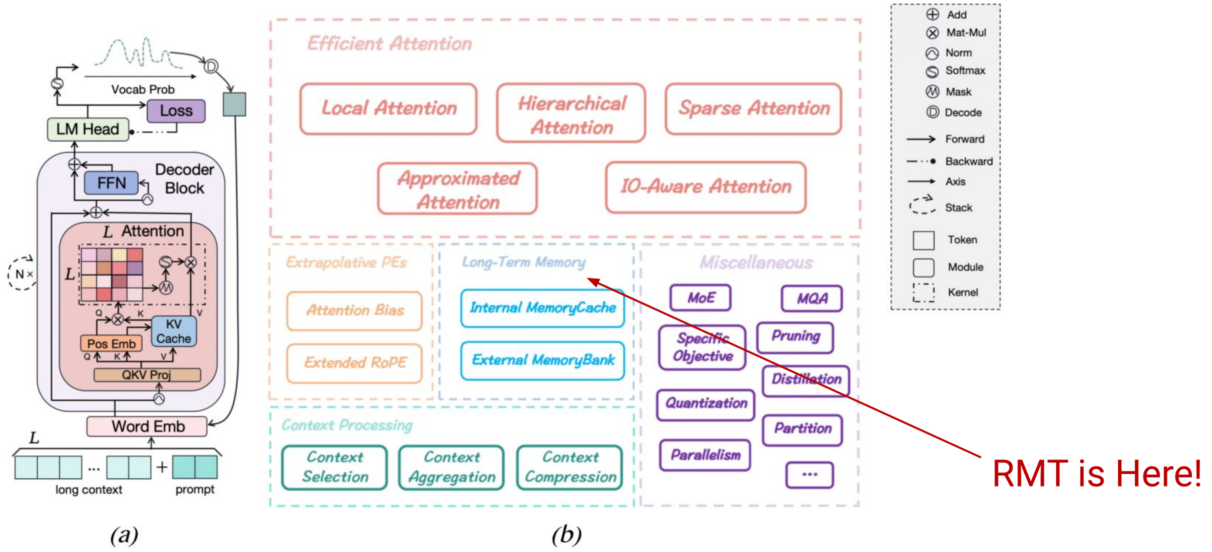


Fig. 1. The overview of the survey: *(a)* The typical architecture anatomy diagram of contemporary Transformer-based decoder-only LLMs, with the legend on the far top right; *(b)* The taxonomy of methodologies for enhancing Transformer architecture modules (corresponding to *(a)* by color): Efficient Attention (submodule of attention kernel), Long-Term Memory (targeting KV cache), Extrapolative PEs (against the positional embedding module), Context Processing (related to context pre/post-processing), and Miscellaneous (general for the whole Decoder Block as well as the Loss module).

Carnegie Mellon University

## Related Work

**Question:**

- To improve the efficiency of attention ➡ we want input length ***s*** to be not that long
- But how to reduce ***s*** while still maintaining the model's ability to understand and generate long contexts?

**Solution:**

- **Idea1:** Making Attention sparse to reduce computation
  - Longformer, Big Bird
- **Idea2:** Making Transformer recurrent to reduce memory usage
  - Transformer-XL, Compressive Transformer, Memory Transformer

## Related work: Make Attention Sparse

**Mistral-7B** (Jiang et al. 2023)

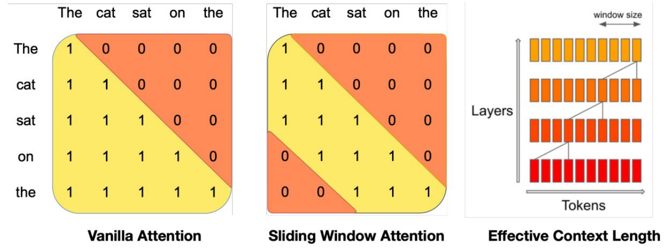
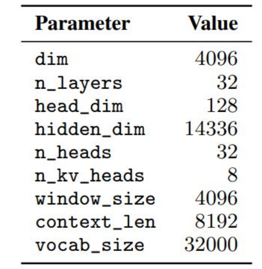


**Figure 1: Sliding Window Attention.** The number of operations in vanilla attention is quadratic in the sequence length, and the memory increases linearly with the number of tokens. At inference time, this incurs higher latency and smaller throughput due to reduced cache availability. To alleviate this issue, we use sliding window attention: each token can attend to at most $W$ tokens from the previous layer (here, $W = 3$). Note that tokens outside the sliding window still influence next word prediction. At each attention layer, information can move forward by $W$ tokens. Hence, after $k$ attention layers, information can move forward by up to $k \times W$ tokens.



| Parameter | Value |
|---|---|
| dim | 4096 |
| n_layers | 32 |
| head_dim | 128 |
| hidden_dim | 14336 |
| n_heads | 32 |
| n_kv_heads | 8 |
| window_size | 4096 |
| context_len | 8192 |
| vocab_size | 32000 |

**Table 1: Model architecture.**

- SWA exploits the stacked layers of a transformer to attend information beyond the window size ***w***
- The hidden state in position ***i*** of the layer ***k***, ***hi*** attends to all hidden states from the previous layer with positions between ***i − w*** and ***i***.
- Recursively, ***hi*** can access tokens from the input layer at a distance of up to ***w × k*** tokens

Mistral 7B (Jiang et al., 2023)

## Related work: Make Attention Sparse

### Longformer (Beltagy et al. 2020)

**Local attention**: local attention is controlled by a sliding window of fixed size ***w***

**Global attention of preselected tokens**: Longformer has a few pre-selected tokens (e.g. [CLS] token) assigned with global attention span, that is, attending to all other tokens in the input sequence.

**Dilated attention**: Dilated sliding window of fixed size ***r*** and gaps of dilation size ***d***

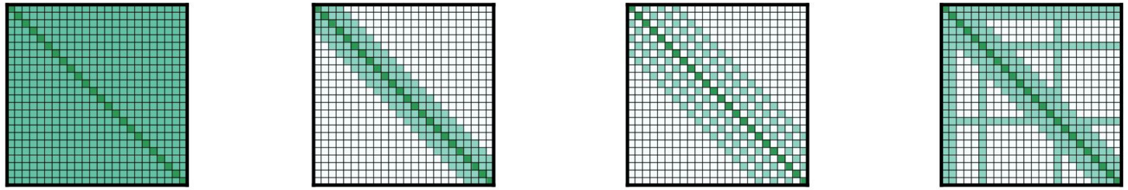

(a) Full $n^2$ attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure 2: Comparing the full self-attention pattern and the configuration of attention patterns in our Longformer.

Longformer: The Long-Document Transformer (Beltagy et al., 2020), Big Bird: Transformers for Longer Sequences (Zaheer at al. 2020)

# Related work: Make Attention Sparse

**Big Bird** (Zaheer et al. 2020)

**Random attention**: Each token uses ***r*** random attention

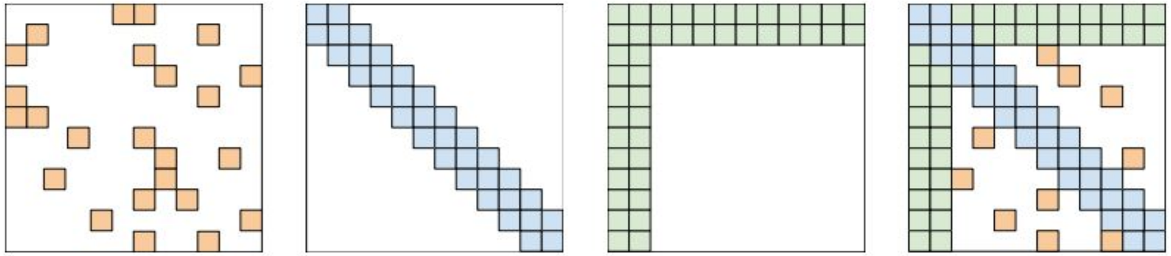


Figure 1: Building blocks of the attention mechanism used in BIGBIRD. White color indicates absence of attention. (a) random attention with $r = 2$, (b) sliding window attention with $w = 3$ (c) global attention with $g = 2$. (d) the combined BIGBIRD model.

## Related work: Make Attention Sparse

**Longformer** (Beltagy et al. 2020) **Big Bird** (Zaheer et al. 2020)

**Limitations**: A common constraint of these methods is that **memory** requirements grow with input size during both training and inference, inevitably limiting input scaling due to hardware constraints.

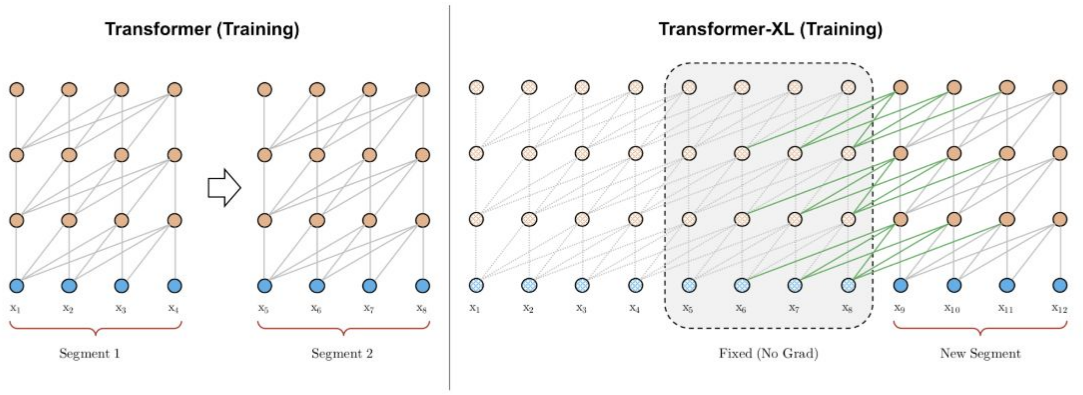
**What can RMT do**: In contrast, recurrent approaches have constant memory complexity during inference.

## Related Work: Transformers with Context Memory

**Main Idea**: Long inputs are divided into smaller segments, processed sequentially with memory to access information from past segments

### Transformer-XL(Dai et al., 2019; "XL" stands for "extra long")

- Makes use of longer context by reusing hidden states between segments
- Keys and values rely on extended hidden states, while queries only consume hidden states at the current step.



$$\widetilde{\mathbf{h}}_{\tau+1}^{(n-1)} = \left[\text{stop-gradient}(\mathbf{h}_{\tau}^{(n-1)}) \circ \mathbf{h}_{\tau+1}^{(n-1)}\right]$$
$$\mathbf{Q}_{\tau+1}^{(n)} = \mathbf{h}_{\tau+1}^{(n-1)} \mathbf{W}^{q}$$
$$\mathbf{K}_{\tau+1}^{(n)} = \widetilde{\mathbf{h}}_{\tau+1}^{(n-1)} \mathbf{W}^{k}$$
$$\mathbf{V}_{\tau+1}^{(n)} = \widetilde{\mathbf{h}}_{\tau+1}^{(n-1)} \mathbf{W}^{v}$$
$$\mathbf{h}_{\tau+1}^{(n)} = \text{transformer-layer}(\mathbf{Q}_{\tau+1}^{(n)}, \mathbf{K}_{\tau+1}^{(n)}, \mathbf{V}_{\tau+1}^{(n)})$$

Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context (Dai et al., 2019)

Carnegie Mellon University

## Related Work: Transformers with Context Memory

**Main Idea**: Long inputs are divided into smaller segments, processed sequentially with memory to access information from past segments

### Compressive Transformer (Rae et al., 2019)

- Extends Transformer-XL by compressing past memories to support longer sequences.

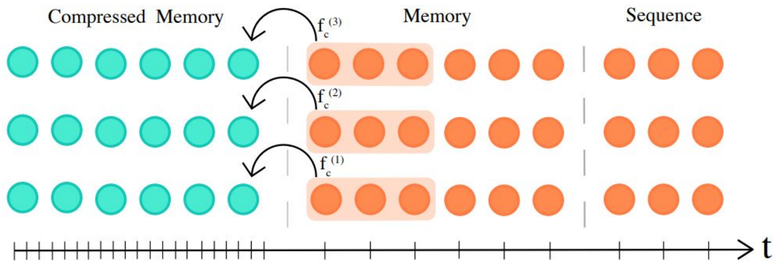


Figure 1: The Compressive Transformer keeps a fine-grained memory of past activations, which are then compressed into coarser *compressed* memories. The above model has three layers, a sequence length $n_s = 3$, memory size $n_m = 6$, compressed memory size $n_{cm} = 6$. The highlighted memories are compacted, with a compression function $f_c$ per layer, to a single compressed memory — instead of being discarded at the next sequence. In this example, the rate of compression $c = 3$.

## Related Work: Transformers with Context Memory

**Compressive Transformer (Rae et al., 2019), Transformer-XL(Dai et al.,)**

- Compressing past memories to support longer sequences.

**Limitations**: A drawback of most existing recurrent methods is the need for architectural modifications that complicate their application to various pre-trained models.

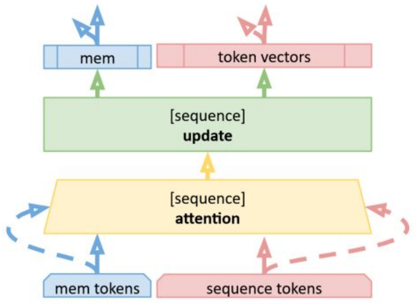
**What can RMT do**: RMT can be built upon any model that uses a common supported interface (like Hugging Face Transformers).

## Related Work: Transformers with Context Memory

**Main Idea**: Long inputs are divided into smaller segments, processed sequentially with memory to access information from past segments

### Memory Transformer(Burtsev et al., 2021)

- Extend the Transformer by adding **[mem]** tokens at the beginning of the input sequence and train the model to use them as **universal memory storage**



$$X^{mem+seq} = [X^{mem}; X^{seq}] \in \mathbb{R}^{(n+m)\times d}, X^{mem} \in \mathbb{R}^{m\times d}, X^{seq} \in \mathbb{R}^{n\times d}.$$

This modification can be applied independently to encoder and/or decoder. The rest of the Transformer stays the same with the multi-head attention layer processing the extended input.

Memory Transformer (Burtsev et al., 2021)

Carnegie Mellon University

## Topic list

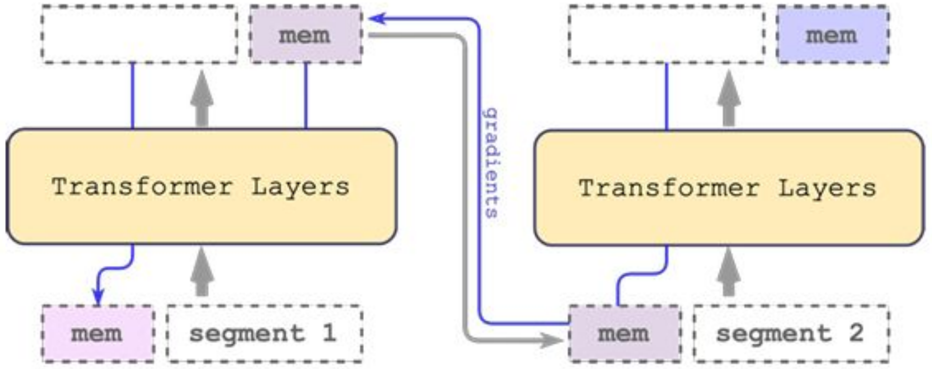
Carnegie Mellon University

## Recurrent Memory Transformer (RMT)

- **What does RMT aim to solve?**
  - Attention Complexity ✔
  - Max-Length Constraint ✔
  - In-Context Memory ✔
- **What is RMT?**
  - Based on special memory tokens similar to **Memory Transformer**, segment-level recurrence as in **Transformer-XL**.
  - **Memory** allows to store and process local and global information as well as to pass information between segments of the long sequence with the help of **recurrence**.

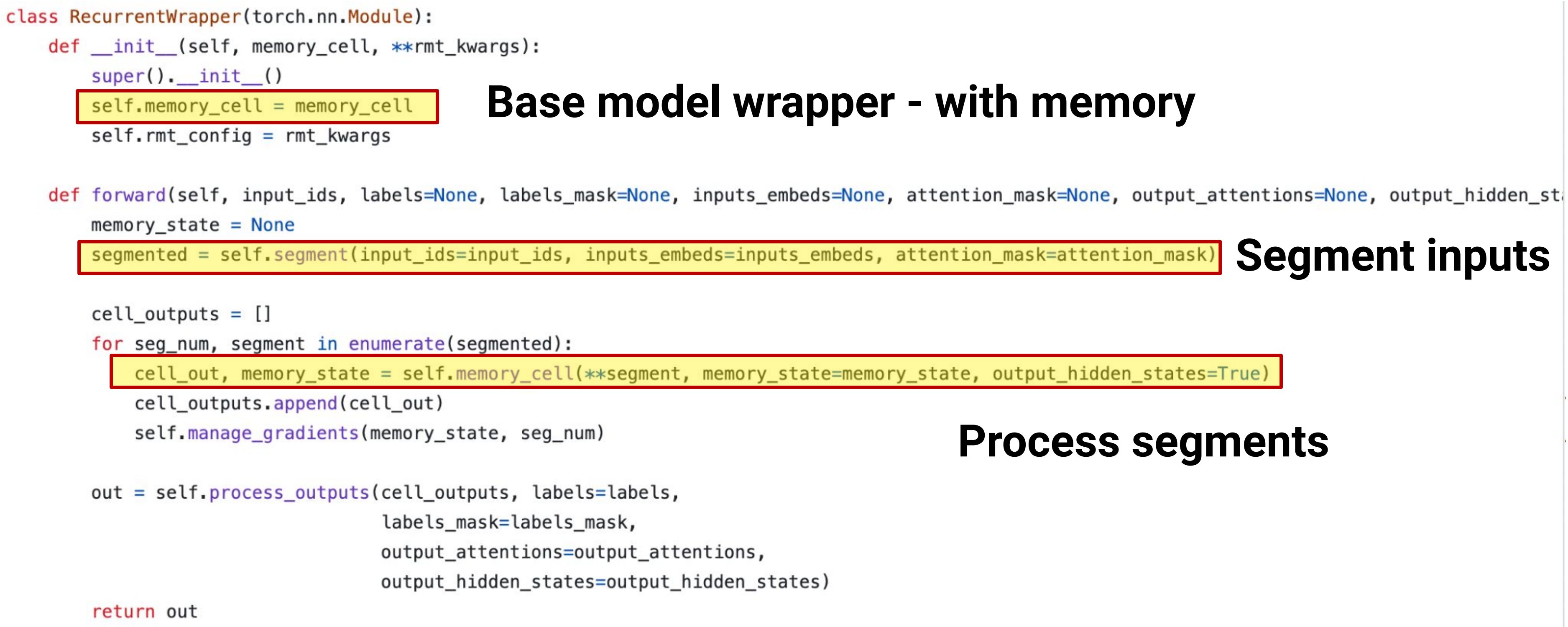
## Recurrent Memory Transformer (RMT)

- **How does it work?**
  - **No changes to Transformer model**
  - Adding **special memory tokens** to the input sequence
  - The model is trained to control both **memory operations** and **sequence representatio**
    processing.



Carnegie Mellon University

## Recurrent Memory Transformer (RMT) - Forward



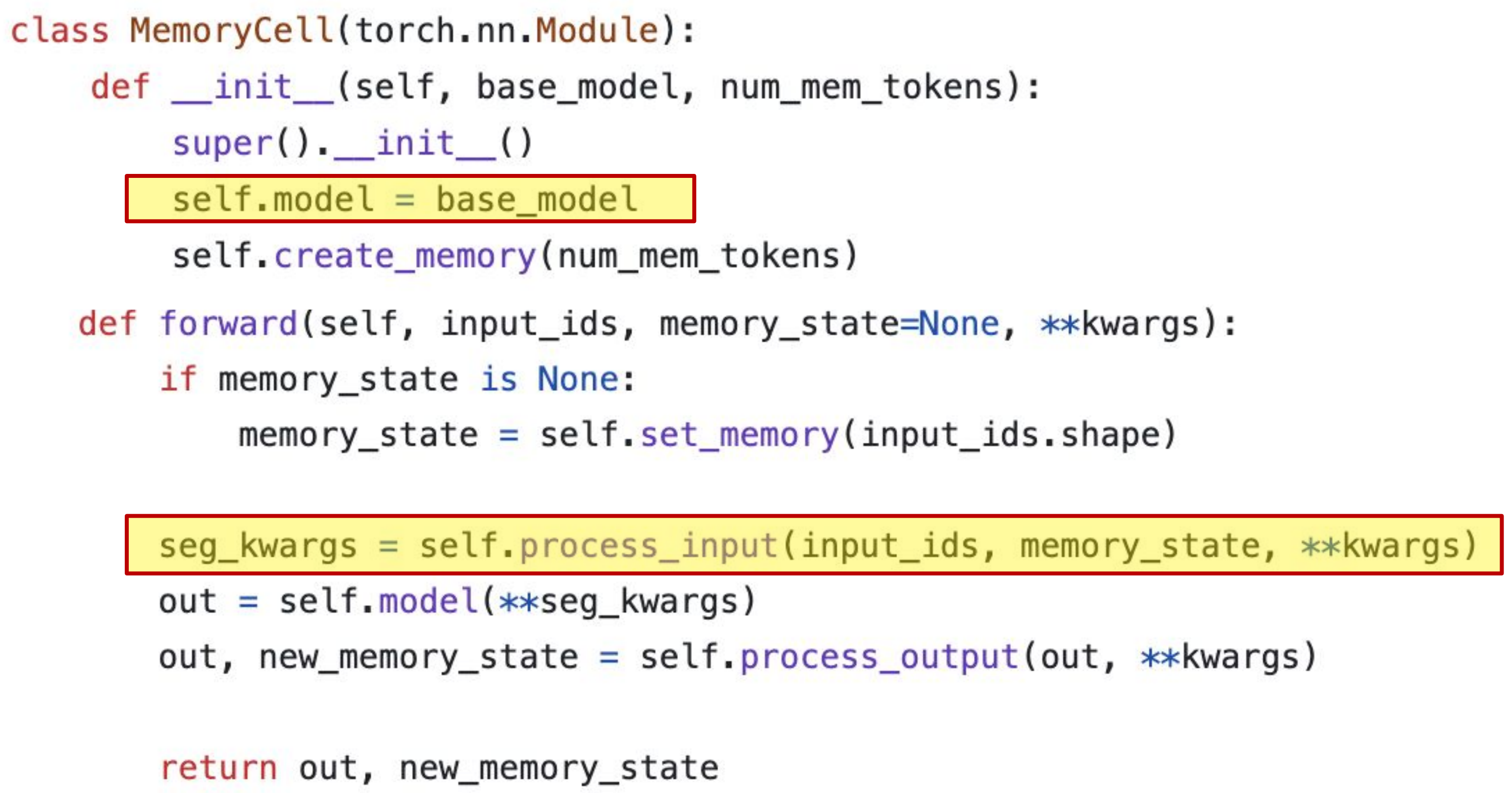
```
class RecurrentWrapper(torch.nn.Module):
    def __init__(self, memory_cell, **rmt_kwargs):
        super().__init__()
        self.memory_cell = memory_cell
        self.rmt_config = rmt_kwargs

    def forward(self, input_ids, labels=None, labels_mask=None, inputs_embeds=None, attention_mask=None, output_attentions=None, output_hidden_st
        memory_state = None
        segmented = self.segment(input_ids=input_ids, inputs_embeds=inputs_embeds, attention_mask=attention_mask)

        cell_outputs = []
        for seg_num, segment in enumerate(segmented):
            cell_out, memory_state = self.memory_cell(**segment, memory_state=memory_state, output_hidden_states=True)
            cell_outputs.append(cell_out)
            self.manage_gradients(memory_state, seg_num)

        out = self.process_outputs(cell_outputs, labels=labels,
                                   labels_mask=labels_mask,
                                   output_attentions=output_attentions,
                                   output_hidden_states=output_hidden_states)
        return out
```

**Base model wrapper - with memory**

**Segment inputs**

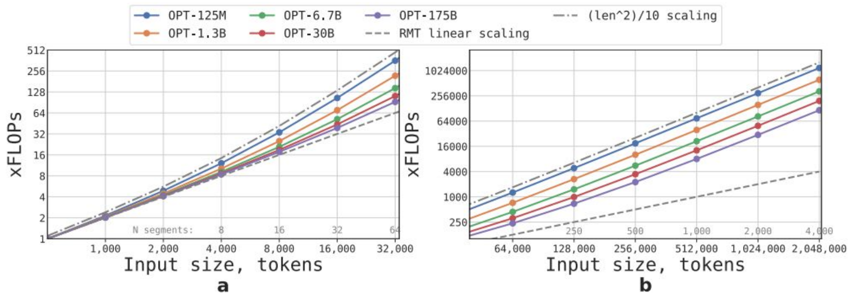
**Process segments**

## Recurrent Memory Transformer (RMT) - Memory Cell



```
class MemoryCell(torch.nn.Module):
    def __init__(self, base_model, num_mem_tokens):
        super().__init__()
        self.model = base_model
        self.create_memory(num_mem_tokens)

    def forward(self, input_ids, memory_state=None, **kwargs):
        if memory_state is None:
            memory_state = self.set_memory(input_ids.shape)

        seg_kwargs = self.process_input(input_ids, memory_state, **kwargs)
        out = self.model(**seg_kwargs)
        out, new_memory_state = self.process_output(out, **kwargs)

        return out, new_memory_state
```

## Recurrent Memory Transformer (RMT)



- RMT inference scales linearly w.r.t. FLOPS for any model size if the segment length is fixed
- Linear scaling is achieved by dividing a input sequence into segments and computing the full attention matrix only within segment boundaries
- RMT requires fewer FLOPs than non-recurrent models for sequence with more than one segment

## Topic list

Carnegie Mellon University

## Memorization Tasks

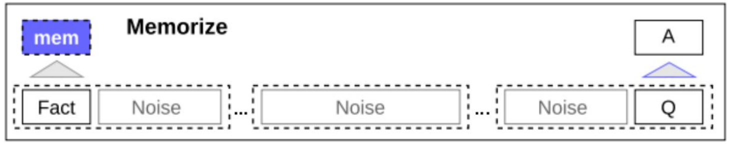
- Goal: We want to see if RMT can memorize “facts” to answer questions in QA
- Example:

  *Fact: Daniel went back to the hallway.*

  *Question: Where is Daniel?*

  *Answer: hallway*

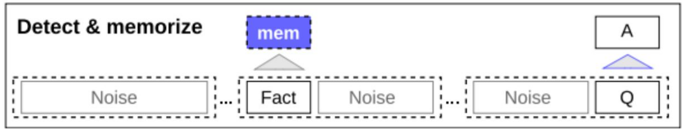
- Task 1: RMT’s ability to write and store information in memory for an extended time.



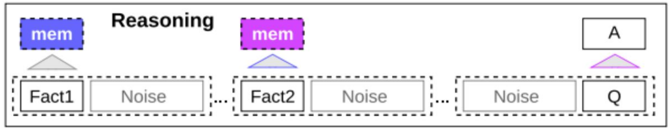
Fact is placed at the start of a sequence

## Memorization Tasks

- Fact detection task: moves the fact to a random position in the input, requiring the model to first distinguish the fact from irrelevant text, write it to memory, and later use it to answer the question.



- Reasoning Task: ability to operate with several facts and current context



*Example:* *Fact1: The hallway is east of the bathroom. Fact2: The bedroom is west of the bathroom.*
*Question: What is the bathroom east of?*
*Answer: bedroom*

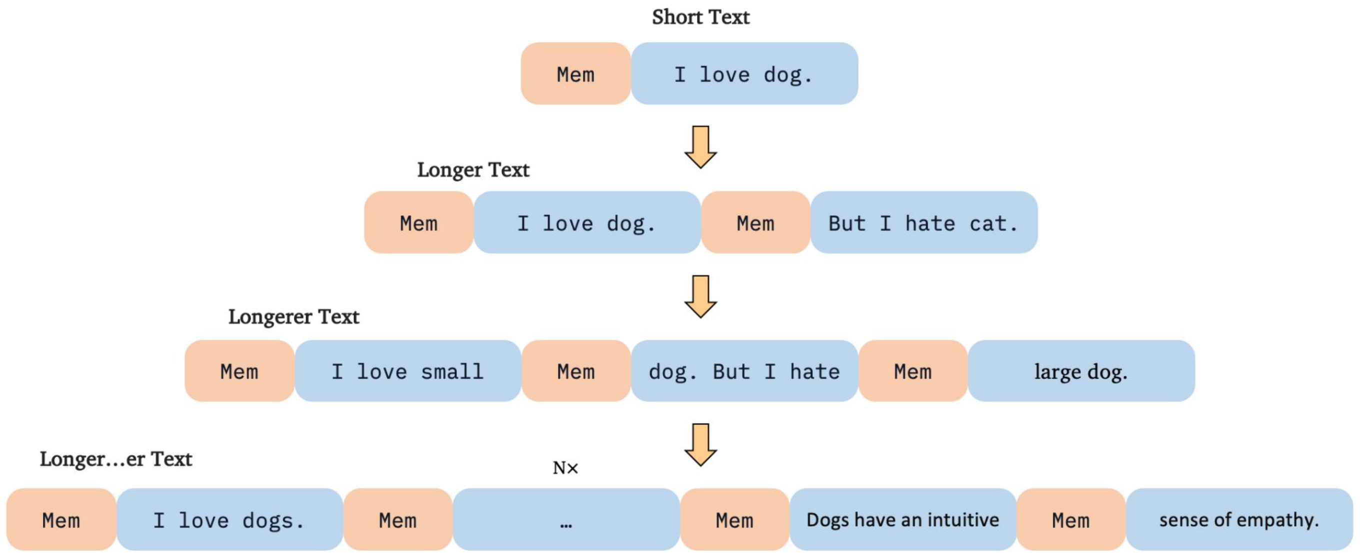
## Topic list

Carnegie Mellon University

## Learning Memory Operations

Question 1: How improve training stability of the original RMT？

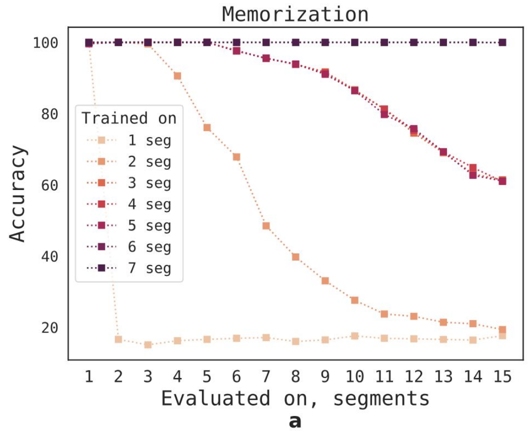
Answer: Curriculum Learning.



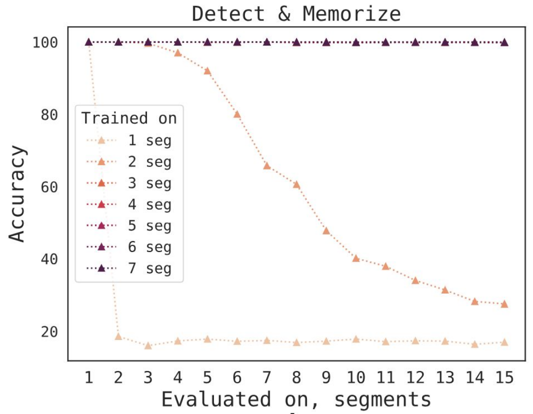
## Learning Memory Operations

Question 2: How well does RMT generalize to different sequence lengths?

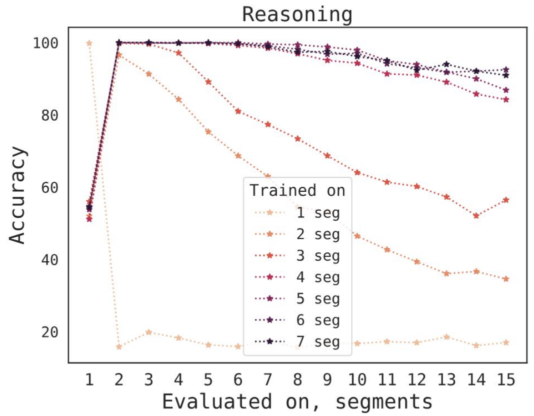
Answer: The generalization ratio enlarges with increasing numbers of training segmentations.



a



b



c

## Learning Memory Operations

Question 2: How well does RMT generalize to different sequence lengths?

Answer: We observe a extrapolation of more than 500 times on a pre-trained BERT model.

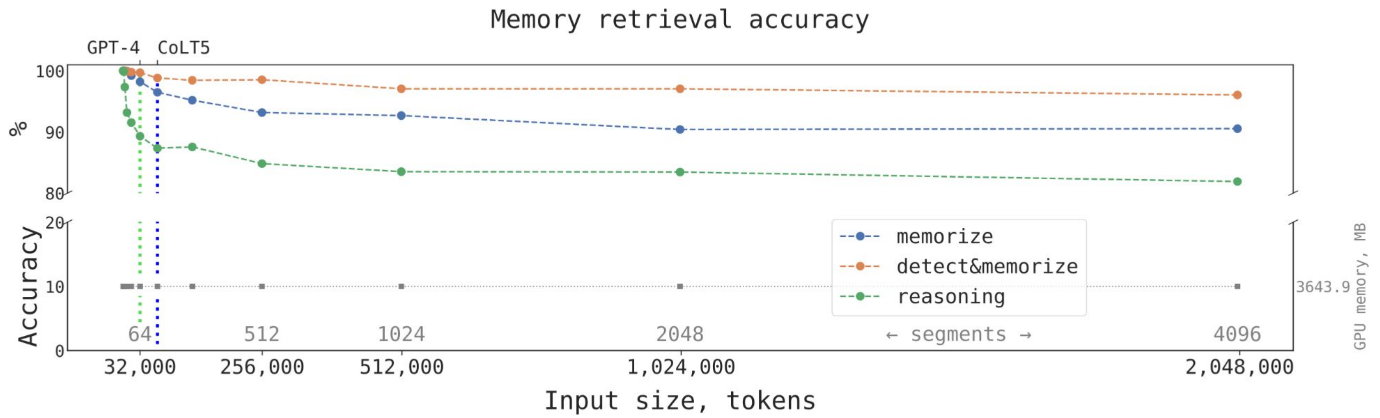
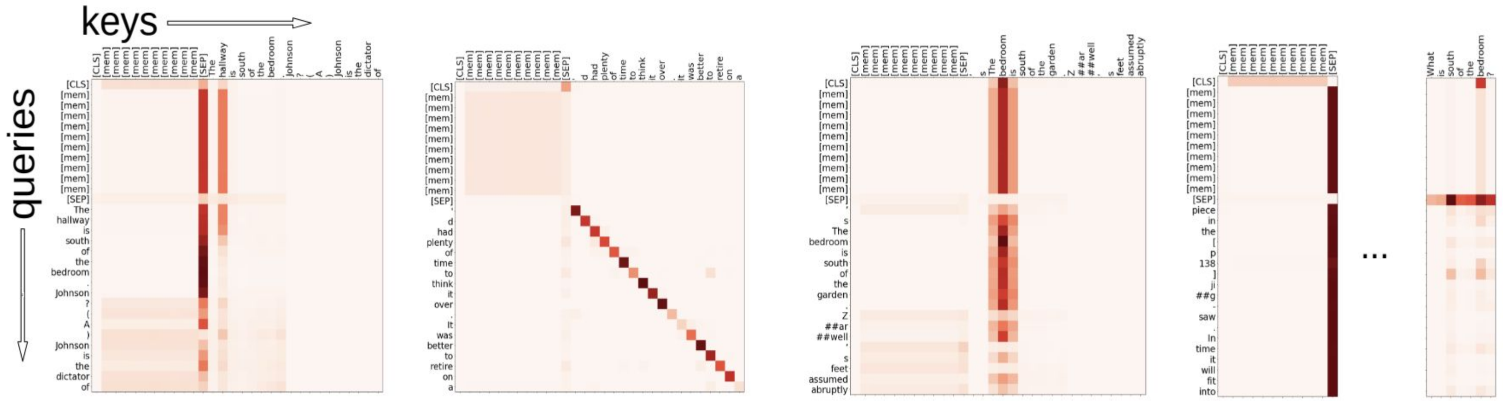


Figure 5: **Recurrent Memory Transformer retains information across up to** $2 \times 10^6$ **tokens**. By augmenting a pre-trained BERT model with recurrent memory (Bulatov, Kuratov, and Burtsev 2022), we enabled it to store task-specific information across 7 segments of 512 tokens each. During inference, the model effectively utilized memory for up to 4,096 segments with a total length of 2,048,000 tokens—significantly exceeding the largest input size reported for transformer models (64K tokens for CoLT5 (Ainslie et al. 2023), and 32K tokens for GPT-4 (OpenAI 2023), and 100K tokens for Claude). This augmentation maintains the base model's memory size at 3.6 GB in our experiments.

## Learning Memory Operations

Question 3: Can RMT identify important facts and store them in memory？



Detect fact 1
write to memory

No fact in input
retain memory

Detect fact 2
write to memory

Use memory
answer the question

Carnegie Mellon University

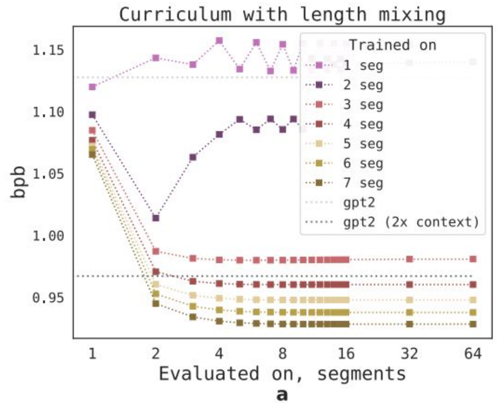
## Topic list

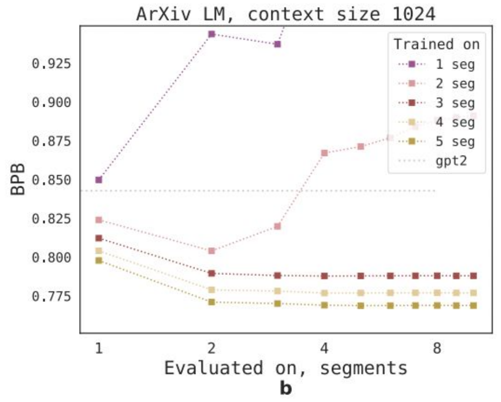
Carnegie Mellon University

# Natural and Formal Language Modelling

- RMT trained for an equal number of steps as the baseline GPT-2 displays substantially lower perplexity values

- Increasing number of segments in training RMT exhibits better tolerance to longer history sizes.

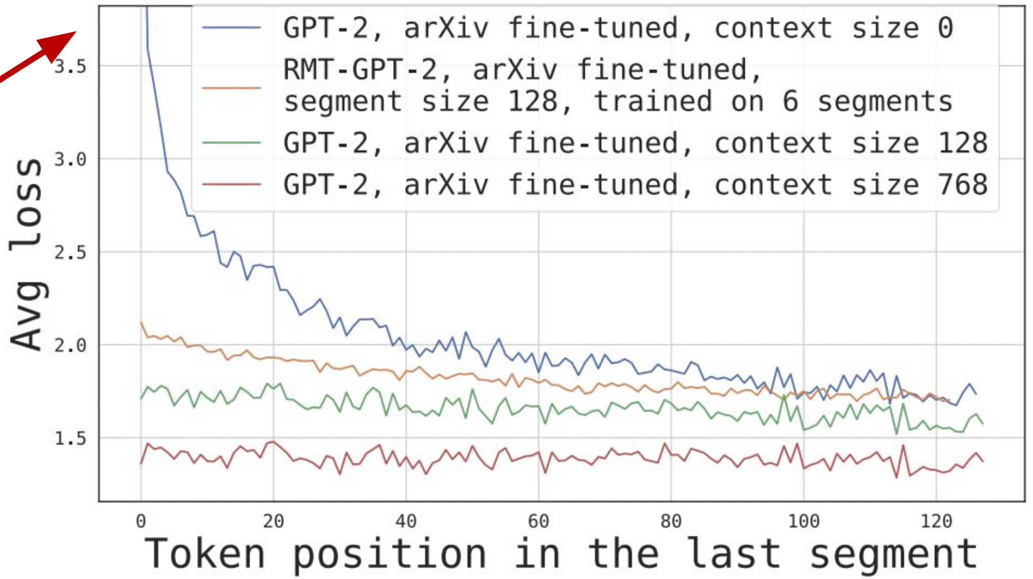




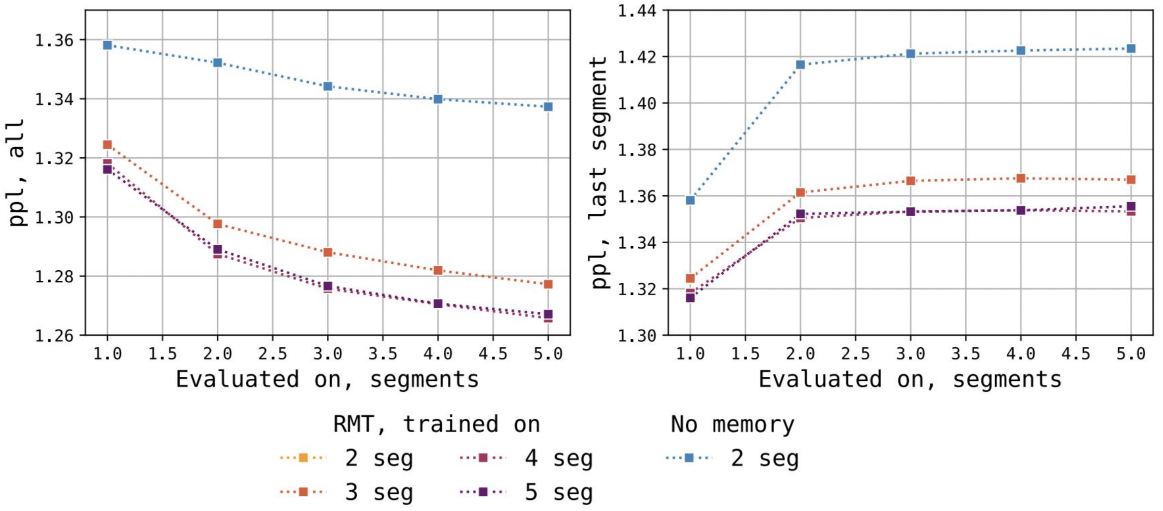
# Natural and Formal Language Modelling

RMT ensures equally good prediction for all tokens due to carryover of information from the previous segment

High initial loss due to lack of context



# Natural and Formal Language Modelling



The RMT model improves perplexity compared to the memory-less model.

However, training with 4 or more segments does not enhance predictions for longer sequences

Carnegie Mellon University

## Topic list

Carnegie Mellon University

## Conclusion

- Problem: Long input scaling in Transformers (Encoder-only & Decoder-only)
- Solution: Segment-level recurrence using recurrent memory (RMT , a kind of compression)
  1. Linear Inference Complexity
  2. Limitation of Sequence Length
  3. In-context Memory
- Training method: Curriculum Learning (short → long → longer → ... → longer)
- Extrapolation: RMT can handle sequences exceeding 1 million tokens while only training on sentence no more than 5k tokens. Therefore, it can maintain computational complexity during training and inference.

Carnegie Mellon University

## Discussion

- RMT perform well in specialized tasks. What about other general tasks?
- Comparing to other recurrent-based approaches, such as Mamba [1] and Griffin [2], what are the advantages and disadvantages of RMT ?
- Evaluated on BERT, Opt and GPT2, the effectiveness of RMT on recent LLMs remains unknown.

[1] Albert Gu, Tri Dao, Mamba: Linear-Time Sequence Modeling with Selective State Spaces.

[2] De *et al.* , Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models.