Block-wise Recurrent Transformer: Enabling Effective Length Extrapolation

Siyuan Liang
siyuan.ethan.liang@gmail.com

Abstract

The Transformer architecture, especially in decoder-only form for autoregressive language modeling, has achieved remarkable success in natural language processing due to its powerful parallel capabilities and attention mechanism. However, the standard Transformer's attention mechanism is stateless, which poses dual challenges of $O(N^2)$ computational complexity and inference memory consumption when processing long texts. Existing solutions such as Transformer-XL introduce segment-level recurrence, but their state transfer is limited to direct copying of the KV cache, lacking deep state evolution, which restricts their theoretical receptive field to $N \times L$ (window length $\times$ number of layers). In this paper, we propose a **Block-wise Recurrent Transformer** architecture (also known as Memory Attention). This architecture transforms the Transformer from a stateless parallel computer into a stateful sequence model by introducing an explicit **Recurrent State**. Our core innovations lie in: (1) **Explicit State Evolution**: introducing a non-linear projection (Projection FFN) between blocks, enabling memory to "think" and "compress" in the temporal dimension, rather than just being passively stored; and (2) **Stateful Segment Training**: combining the classic TBPTT (Truncated Backpropagation Through Time) idea from RNNs, maintaining state transfer across batches during training, enabling the model to learn true long-distance dependencies rather than just sliding windows. Experimental results show that the model, trained on a length of only 256, can generalize to sequences of 4096 or even longer, with loss continuously decreasing as length increases (Train Short, Test Long), demonstrating effective length extrapolation. Code and demo are available at `https://github.com/siyuanseever/llama2Rnn`.

1 Introduction

> "Language is high-level intelligence, while physical space is real life. But time is the dimension that gives meaning to both."

Current large language models (LLMs) excel at modeling information within a finite attention window, but they remain limited when processing ultra-long sequences. Standard Transformers effectively rely on a large context window to maintain short-term dependencies.

However, robust sequence understanding requires **time awareness**. Humans do not store every token verbatim while reading a book; instead, they continuously update a compact "state of understanding" over time.

The **Block-wise Recurrent Transformer** proposed in this paper aims to give "time" back to the model. Rather than indefinitely expanding the context window, we focus on **stateful modeling with unbounded effective history**.

Preprint. Under review.

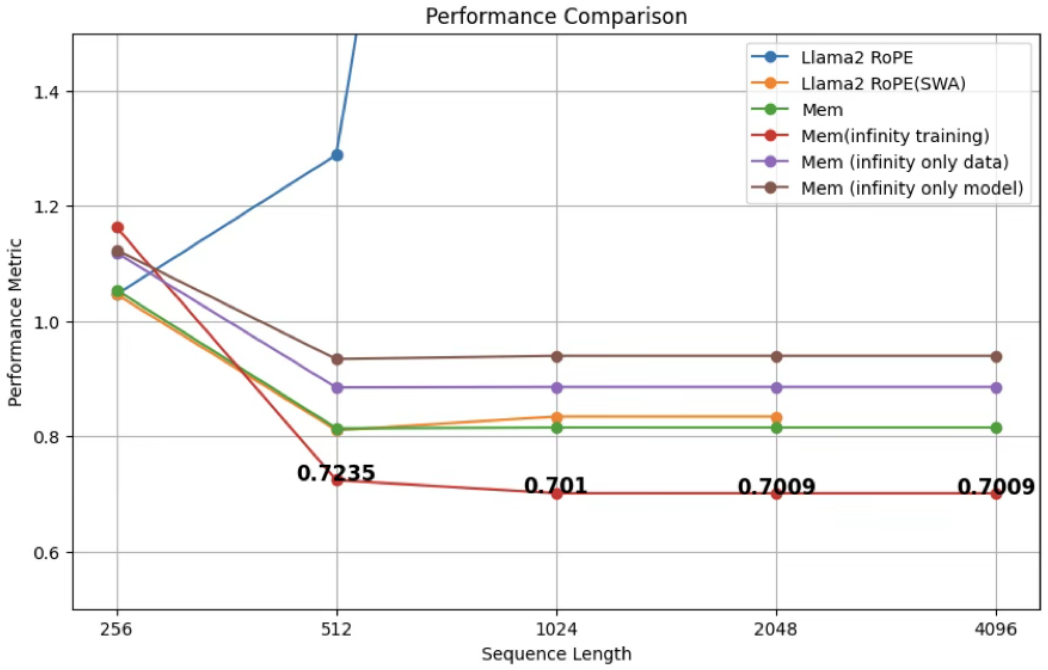

Figure 1: Experimental results visualization. Our method exhibits consistent "Train Short, Test Long" gains as the evaluation length increases.

2 Related Work

2.1 Transformer-XL and Receptive Field Limitations

Transformer-XL [2] first introduced Segment-level Recurrence, extending context by caching the KV state of the previous segment. However, its state transfer is a linear hierarchical transfer. The token at layer i can only see the previous segment of layer $i-1$. This means that to see information from K segments ago, the signal must pass down through K layers. Therefore, its maximum receptive field is physically limited to $N \times L$. For very long sequences, this limitation remains a bottleneck.

2.2 RNN and Linear Attention

RNNs (such as LSTM [5]) and Linear Attention (such as Linear Transformer [6], Mamba [4]) solve the length problem through $O(1)$ inference state. However, they often lose high-precision local information during the "compression" process, performing worse than standard Attention on tasks requiring precise recall (Copy Tasks).

2.3 Our Method: Block-wise Recurrence

Our method sits in the "Sweet Spot" between the two:

- **Within chunk**: retain standard $O(C^2)$ attention to preserve high-precision local information.
- **Across chunks**: use a recurrent state to propagate information over time, yielding a theoretically unbounded effective context.

Standard Transformers face two core challenges when extrapolating beyond the training length:

1. **Position extrapolation**: for position encodings such as RoPE, relative position patterns beyond the training range fall into unseen regions, which can degrade attention behavior.

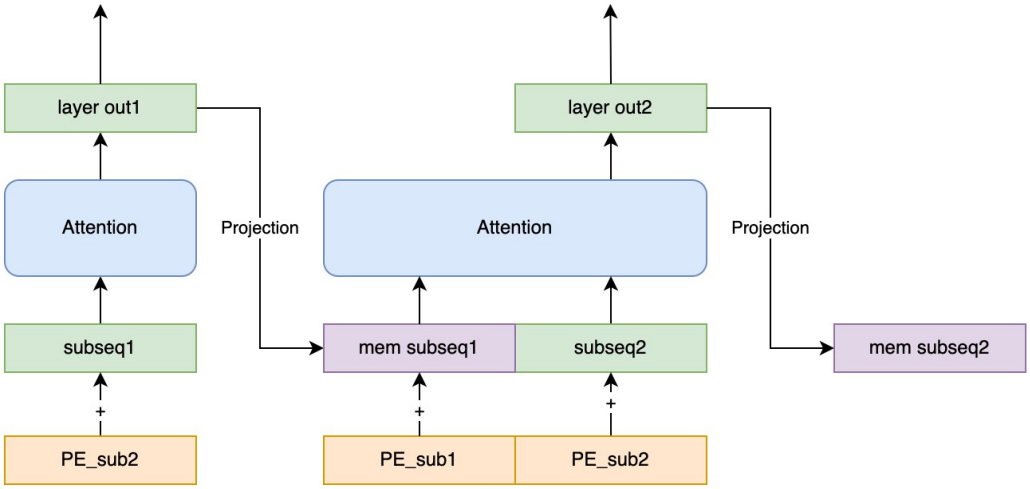

Figure 2: Memory Attention Architecture.

2. **Attention dilution**: as the sequence length grows, the attention distribution can become increasingly diffuse, making it harder to focus on key information.

Block-wise recurrence mitigates both issues by construction:

- **Mitigating position extrapolation**: each block has a fixed length (e.g., 256), so the model only needs to encode relative positions within a bounded range; increasing the total sequence length does not enlarge the within-block positional range.
- **Mitigating attention dilution**: each attention computation is bounded by the block size, keeping the effective attention scale and density stable regardless of the total sequence length.

2.4 Comparative Analysis

We compared the Block-wise Recurrent Transformer with two other major long-context architectures:

- **vs. Recurrent Memory Transformer (RMT)**: RMT [1] uses a "Memory as Tokens" mechanism, splicing memory as special tokens into the input sequence. Its memory update relies entirely on shared Self-Attention weights. In contrast, our method maintains a **Latent State independent of the token sequence**, and performs explicit state evolution through a dedicated Projection FFN. This decoupled design makes the memory forgetting and update mechanism more independent and flexible, not completely constrained by the attention distribution between tokens.
- **vs. Compressive Transformer**: DeepMind's Compressive Transformer uses a "Compression" strategy to compress the KV Cache sliding out of the window into a more compact vector. This is essentially still a **Retrieval-based** mechanism, where the model needs to "look back" at the compressed history. Our method belongs to a **Recurrent-based** mechanism, where historical information is continuously integrated into the current state vector (Summary State). The model only needs to focus on the current state without backtracking history, theoretically achieving $O(1)$ inference space complexity (relative to history length).

3 Methodology

Our model core lies in replacing the standard Attention layer with a **Memory Attention** layer.

3.1 State Definition & Evolution

We split the input sequence X into chunks of length L: $X_1, X_2, ..., X_T$. For the current chunk X_t, we take the **output of the same layer at the previous moment** O_{t-1} as the original memory state. Before injecting it into the current calculation, we first perform a non-linear mapping (Memory Projection) on it:

$$M_{t-1} = O_{t-1} \quad \text{(Output from previous chunk)} \tag{1}$$

$$M'_{t-1} = \text{Norm}(M_{t-1} + \text{FFN}(M_{t-1})) \tag{2}$$

3.2 Memory Injection

The evolved memory M'_{t-1} is projected into the Query-Key-Value space and spliced as a prefix to the Attention calculation of the current chunk. Notably, although Q, K, V all have the memory part spliced, we only keep the latter half of the Attention output (i.e., the part corresponding to the current chunk X_t) as the input for the next layer and the new memory O_t:

$$Q = [W_{qm}M'_{t-1}; W_q X_t], \quad K = [W_{km}M'_{t-1}; W_k X_t], \quad V = [W_{vm}M'_{t-1}; W_v X_t] \tag{3}$$

$$\text{AttnOutput} = \text{Softmax}(\frac{QK^T}{\sqrt{d}} + \text{RoPE})V \tag{4}$$

$$O_t = \text{AttnOutput}[:, \text{Len}(M) :] \quad \text{(Discard memory part, keep current chunk)} \tag{5}$$

Finally, this truncated O_t serves both as the output of the current layer passed to the next layer and as the new memory M_t passed to the next time step.

This step is the fundamental difference from Transformer-XL. Transformer-XL simply copies the past KV, while we allow the model to **process and reorganize** memory between time steps. This is similar to $h_t = f(h_{t-1}, x_t)$ in RNNs, endowing the model with dynamic forgetting and reinforcement capabilities. More importantly, Transformer-XL's memory transfer is based on **caching of original input features**, with a receptive field limited by layer stacking ($N \times L$); our method, through **non-linear mapping of output features**, compresses historical information into a fixed-size state vector, theoretically achieving **Infinite Temporal Receptive Field**, just as RNNs can capture dependencies of arbitrary length through state recurrence.

3.3 Algorithm Pseudocode

Algorithm 1 Block-wise Recurrent Transformer (Memory Attention)

Require: Input sequence X of shape [Batch, SeqLen, Dim]
Require: Initial memory state M_{init} (optional)
Require: ChunkSize: Length of each processing block
Ensure: Output sequence Y, Final memory state M_{final}
Initialize $M = M_{init}$ if provided else LearnableParameter
Split X into chunks: $X_1, X_2, ..., X_T$
Initialize Output List $Y_{list} = []$
for $t = 1$ to T **do**
    // Standard Projection for Current Chunk
    $Q_x, K_x, V_x = \text{Linear\_QKV}(X_t)$
    // State Evolution (The "Recurrent" Step)
    $M_{evolved} = \text{Norm}(M + \text{FFN}(M))$
    // Memory Injection
    $Q_m, K_m, V_m = \text{Linear\_Mem\_QKV}(M_{evolved})$
    // Concatenate (Memory acts as Prefix)
    $Q = \text{Concat}([Q_m, Q_x], \text{dim=Seq})$
    $K = \text{Concat}([K_m, K_x], \text{dim=Seq})$
    $V = \text{Concat}([V_m, V_x], \text{dim=Seq})$
    // Attention with RoPE
    // Note: RoPE is applied to the full concatenated sequence
    $\text{Output\_Attn} = \text{Attention}(\text{RoPE}(Q), \text{RoPE}(K), V)$
    // Extract Output & Update State
    // Discard memory part, keep only chunk part
    $O_t = \text{Output\_Attn}[:, \text{Length}(M) :]$
    // The output of current block becomes memory for next block
    $M = O_t.\text{detach}()$ if Training_Stateful else O_t
    Append O_t to Y_{list}
end for
$Y = \text{Concat}(Y_{list}, \text{dim=Seq})$
return Y, M

4 Training Strategy: Stateful Segment Training

Structure alone is not enough. If the state is cleared for every batch during training, the model cannot learn to use the state. We introduce **Stateful Segment Training** (essentially the application of RNN's TBPTT technique at the Block level):

1. **State Persistence**: The final state M_i at the end of Batch i is not discarded, but used as the initial state for Batch $i + 1$ after detaching gradients.
2. **Data Continuity**: Ensure that the data of Batch $i + 1$ semantically follows Batch i.

This training method forces the model to treat M as a container for long-term context, thereby breaking the limitation of training length (e.g., 256) and achieving generalization to infinite-length sequences.

5 Experiments

5.1 Length Extrapolation

We validated on the TinyStories [3] dataset. The model only saw a window of length 256 during training, but extrapolated to length 4096 during testing. We compared our method with standard Llama2 (RoPE) [7] and Slide Window Attention. To fairly compare performance at different lengths, we **only calculate the average Loss of the last 256 Tokens of the sequence**. This ensures that

regardless of the inference length, the evaluation metric always focuses on the model's ability to utilize "ultra-long history".

Table 1: Length Extrapolation Performance (Loss on last 256 tokens)

Sequence Length	Standard Llama2	Slide Window	**Ours**
256 (Train)	1.04	1.04	1.05
512	1.28 (Fail)	0.81	**0.72**
1024	3.60	0.83	**0.70**
2048	5.25	0.83	**0.70**
4096	OOM	OOM	**0.70**

Results show:

1. **Standard Llama2** Loss rises rapidly after exceeding training length, suffering severe **length extrapolation collapse**.
2. **Slide Window Attention** avoids collapse, but Loss stagnates around 0.83, indicating it can only use fixed-length local context and cannot benefit from longer history.
3. **Recurrent Transformer** demonstrates significant **Train Short, Test Long** capability. As inference length increases from 512 to 1024, Loss further decreases ($0.72 \rightarrow 0.70$) and stabilizes at 0.70, proving the model successfully compressed historical information into the Recurrent State and effectively utilized these long-term dependencies.

5.2 Training Strategy Experiments

To verify the effectiveness of Stateful Segment Training, we conducted detailed ablation studies. We compared the following settings:

1. **Mem**: Use Memory Attention structure only, without Stateful Training (state cleared between Batches).
2. **Mem (Infinity Only Data)**: Keep data continuity only, but clear state between Batches.
3. **Mem (Infinity Only Model)**: Keep state transfer between Batches only, but data is discontinuous (random shuffle).
4. **Mem (Full Stateful Training)**: Keep both state transfer and data continuity (the complete method proposed in this paper).

Table 2: Ablation Study on Training Strategies

Method	256	512	1024	2048	4096
Mem (No Stateful)	1.0540	0.8138	0.8153	0.8153	0.8153
Mem (Infinity Only Data)	1.1189	0.8845	0.8853	0.8853	0.8853
Mem (Infinity Only Model)	1.1238	0.9338	0.9394	0.9394	0.9394
Mem (Full Stateful)	1.1631	**0.7235**	**0.7010**	**0.7009**	**0.7009**

Conclusion:

- Without Stateful Training (Mem), the model degenerates to an effect similar to Slide Window Attention (Loss stays around 0.81), unable to gain long-distance benefits.
- Data continuity alone or state transfer alone is insufficient for the model to learn long-distance dependencies.
- Only when **both** data continuity and state transfer are satisfied (Full Stateful) can the model truly understand that "the state of the previous Batch corresponds to the end of the previous Batch", thereby achieving a significant decrease in Loss ($0.81 \rightarrow 0.70$).

Results show that our model not only avoids long-distance generalization collapse but also achieves further Loss reduction as context length increases. This proves that the model truly learned to use Recurrent State to store long-term information.

6 Conclusion

Recurrent Transformer is an attempt to return to the essence of sequence modeling. Through simple Block-wise Recurrence and state evolution design, we successfully returned the time dimension to the Transformer. This not only solves the extrapolation problem but also provides a possible path for future LLMs to possess continuous life experience and long-term memory.

References

[1] Aydar Bulatov, Yuri Kuratov, and Mikhail Burtsev. Recurrent memory transformer. In *Advances in Neural Information Processing Systems*, volume 35, pages 11079–11091, 2022.

[2] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, 2019.

[3] Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english? *arXiv preprint arXiv:2305.07759*, 2023.

[4] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[6] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

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