WHY DOES THE EFFECTIVE CONTEXT LENGTH OF LLMS FALL SHORT?

Chenxin An[1] * **Jun Zhang**[2] **Ming Zhong**[3] **Lei Li**[1] **Shansan Gong**[1]
Yao Luo[2] **Jingjing Xu**[2] **Lingpeng Kong**[1]
[1]The University of Hong Kong [2]ByteDance Inc. [3]University of Illinois Urbana-Champaign
https://github.com/HKUNLP/STRING

Gus Fridell, Jason Zuo
10/14/2025 CS 601.771

Motivation

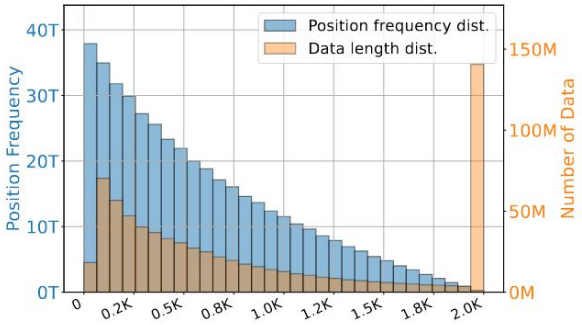
- There is a discrepancy between theoretical improvements and observed performance in context length
- LLMs demonstrate effective context lengths <50% of training length
- 2 main reasons:
  - undertraining of long distance position indices during pre-training and post-training (left skewness)
  - Compounded by inherent difficulty in modeling long-range dependencies

RULER: What's the Real Context Size of Your Long-Context Language Models?

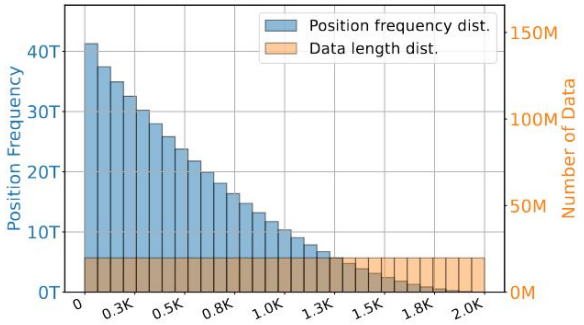
Hsieh, et al., 2024.

Models	Claimed Length	Effective Length	4K	8K	16K	32K	64K	128K	Avg.
Llama2 (7B)	4K	-	85.6	Eff. Length := max(length) >= Llama2(7B) 4K					
Gemini-1.5-Pro	1M	>128K	96.7	95.8	96.0	95.9	95.9	94.4	95.8
GPT-4	128K	64K	96.6	96.3	95.2	93.2	87.0	81.2	91.6
Llama3.1 (70B)	128K	64K	96.5	95.8	95.4	94.8	88.4	66.6	89.6
Qwen2 (72B)	128K	32K	96.9	96.1	94.9	94.1	79.8	53.7	85.9
Command-R-plus (104B)	128K	32K	95.6	95.2	94.2	92.0	84.3	63.1	87.4
GLM4 (9B)	1M	64K	94.7	92.8	92.1	89.9	86.7	83.1	89.9
Llama3.1 (8B)	128K	32K	95.5	93.8	91.6	87.4	84.7	77.0	88.3
GradientAI/Llama3 (70B)	1M	16K	95.1	94.4	90.8	85.4	80.9	72.1	86.5
Mixtral-8x22B (39B/141B)	64K	32K	95.6	94.9	93.4	90.9	84.7	31.7	81.9
Yi (34B)	200K	32K	93.3	92.2	91.3	87.5	83.2	77.3	87.5
Phi3-medium (14B)	128K	32K	93.3	93.2	91.1	86.8	78.6	46.1	81.5
Mistral-v0.2 (7B)	32K	16K	93.6	91.2	87.2	75.4	49.0	13.8	68.4
LWM (7B)	1M	<4K	82.3	78.4	73.7	69.1	68.1	65.0	72.8
DBRX (36B/132B)	32K	8K	95.1	93.8	83.6	63.1	2.4	0.0	56.3
Together (7B)	32K	4K	88.2	81.1	69.4	63.0	0.0	0.0	50.3
LongChat (7B)	32K	<4K	84.7	79.9	70.8	59.3	0.0	0.0	49.1
LongAlpaca (13B)	32K	<4K	60.6	57.0	56.6	43.6	0.0	0.0	36.3

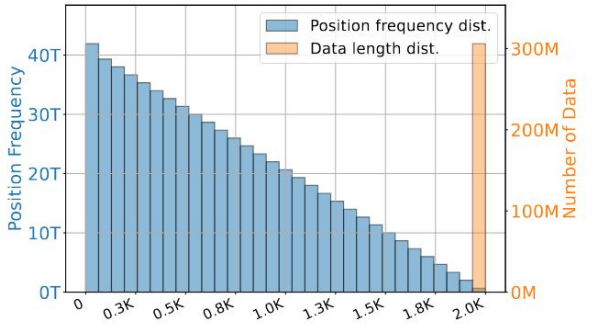
Problem: Left-Skewed Position Frequency Distribution



(a) Natural data distribution



(b) Uniform data distribution



(c) Concatenated data distribution

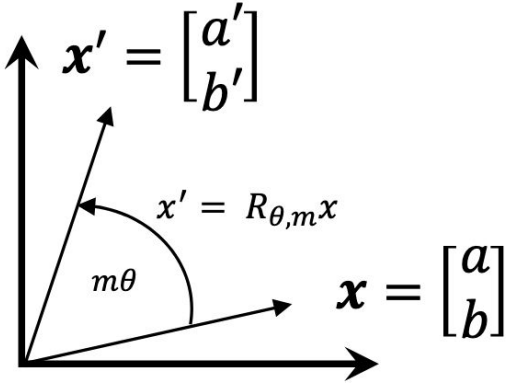
$$f(i) = \sum_{s \subseteq C} \max\big(|s| - i, 0\big), \quad 0 \le i < L$$

"How many times do we compute attention between tokens separated by distance i?"

Intuition: Properties of RoPE

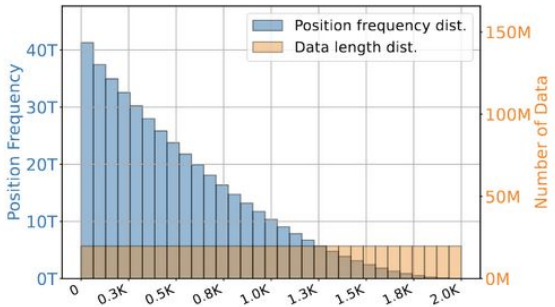
- for queries and keys, $q_i = R_i \cdot q$ (rotate query with rotation matrix R_i)
- $k_□ = R_□ \cdot k$ (rotate key with rotation matrix $R_□$)
- attention score: $q_i^T * k_□ = (R_i \cdot q)^T \cdot (R_□ \cdot k) = q^T \cdot R_i^T \cdot R_□ \cdot k = q^T R_{j-i} k$
  - rotation matrices are orthogonal: $R_i^T = R_i^{-1}$, the inverse rotation "undoes" the rotation
  - R_i rotates q by angle of $i\Theta$, R_i^T rotates by angle of $-i\Theta$
  - rotations compose additively: $R_i^T \cdot R_□ = R_{-i+j}$

$$R_{\theta,m} = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix}$$



Intuition: Building the Position Matrix

- Q^TK gives L x L matrix of attention scores
- P[m][n]:
  - row m = query token at position m
  - col n = key token at position n
  - value = relative position between them = m - n



(b) Uniform data distribution

Key Positions (n)

Query Positions (m)

$$\begin{bmatrix} 0 & & & & \\ 1 & 0 & & & \\ 2 & 1 & 0 & & \\ 3 & 2 & 1 & 0 & \\ 4 & 3 & 2 & 1 & 0 \end{bmatrix}$$

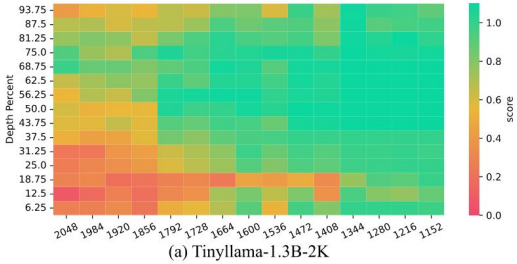
← query 0 attending to keys

← query 1 attending to keys

Challenges

1. Effective-vs-training context length gap is still a problem for models that use RoPE
2. Position frequency imbalance problem
    a. Toeplitz matrix structure is fundamental to relative PE + attention
    b. even with perfect data, we still have linear decay
3. Continual training is cost-ineffective
    a. Llama3.1 used 6 stages of continual training with 800B tokens, you can’t do this
4. Interpolation and Extrapolation approaches fail for the same reasons
    a. Extrapolation (YaRN, NTK-RoPE, etc) train on length L, test on length > L
    b. Problem: if we make position 200K behave like position 100K, what happens if position 100k is already undertrained?
    c. Standard interpolation shares this problem: tail positions are effectively OOD

A PROBING EXPERIMENT



(a) Tinyllama-1.3B-2K

Question:

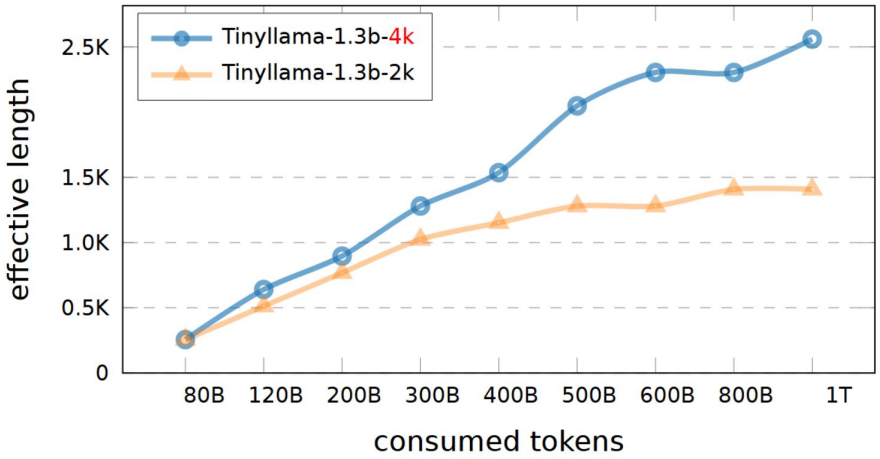
How does position frequency observed in the training corpus and the model's effective length relates?

Setup

- Training
  - 1.3B models: max length of **2k** and **4k**
  - SlimPajama (1T tokens)
- Position frequency is difficult to control directly, so instead:
  - **Consumed tokens**
  - **Training context window size**

Observations

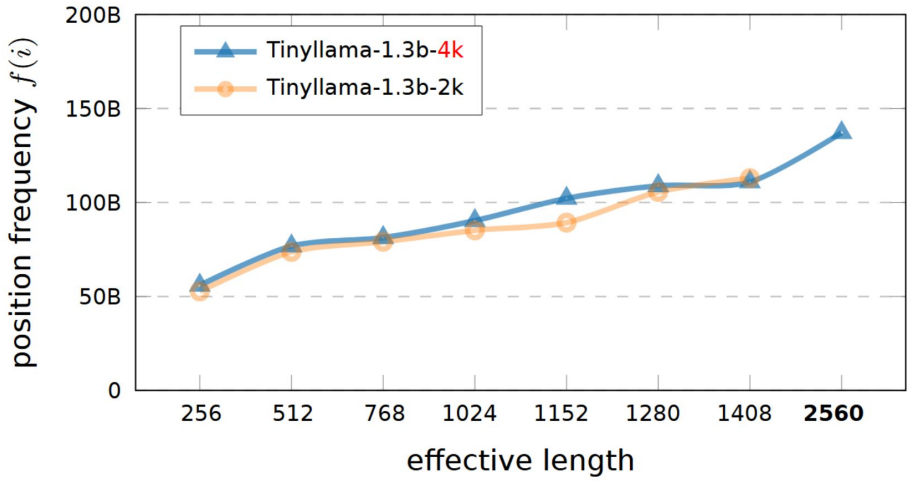
1. Larger training context window consumes fewer tokens to achieve the same effective context length. (Sounds intuitive)



(a) Effective length vs. consumed tokens

Observations

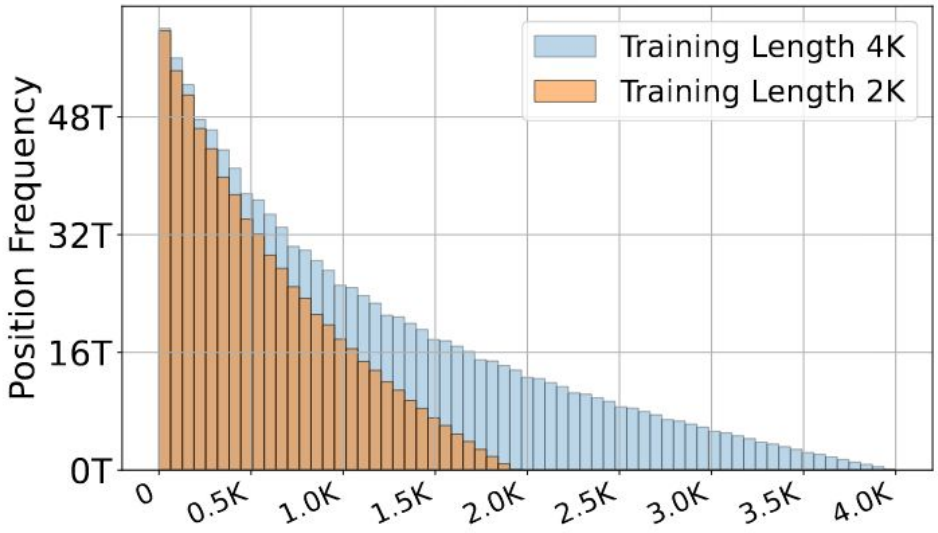
2. Models can achieve similar effective context lengths if they have been exposed to similar frequencies of position indices, even if their maximum training lengths differ. (**The true underlying driver for effective length** 🔑)



(b) Effective length vs. position frequency

Observations

3. The growth trend of the model's effective length aligns with the position frequency distribution: **4k has higher frequency at distancing positions**.

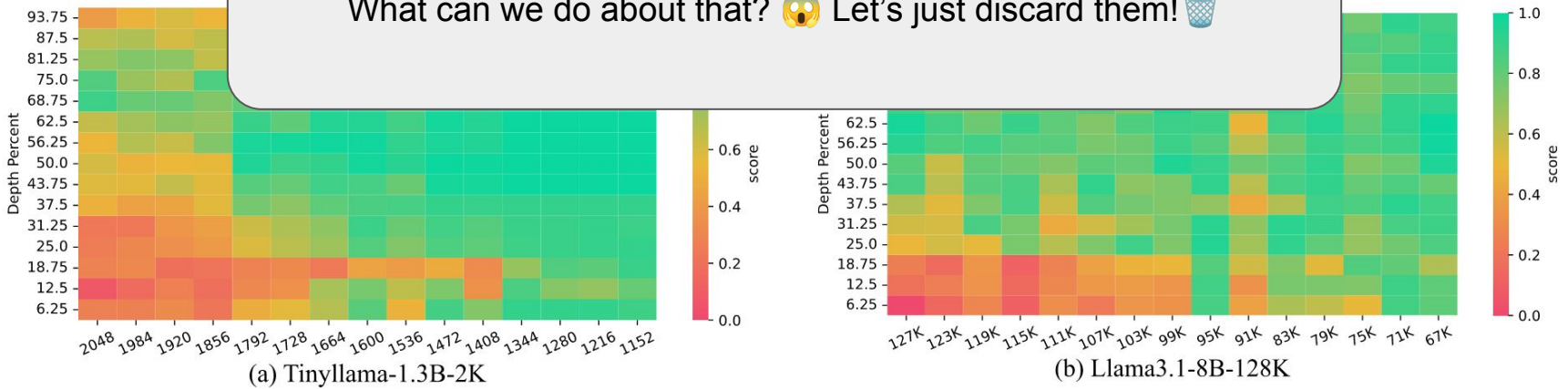


Observations

Bonus. TinyLlama struggles to gather information when the distance exceeds 1,536 tokens. Most failure cases occur within the first L/3 of the document. This may indicate that:

The **last L/3 positions** of current LLMs all **fall in the tail of the position frequency d**

What can we do about that? 😱 Let's just discard them!🗑️



STRING: ShifTed Rotary position embeddING

Key Takeaways from Observations:

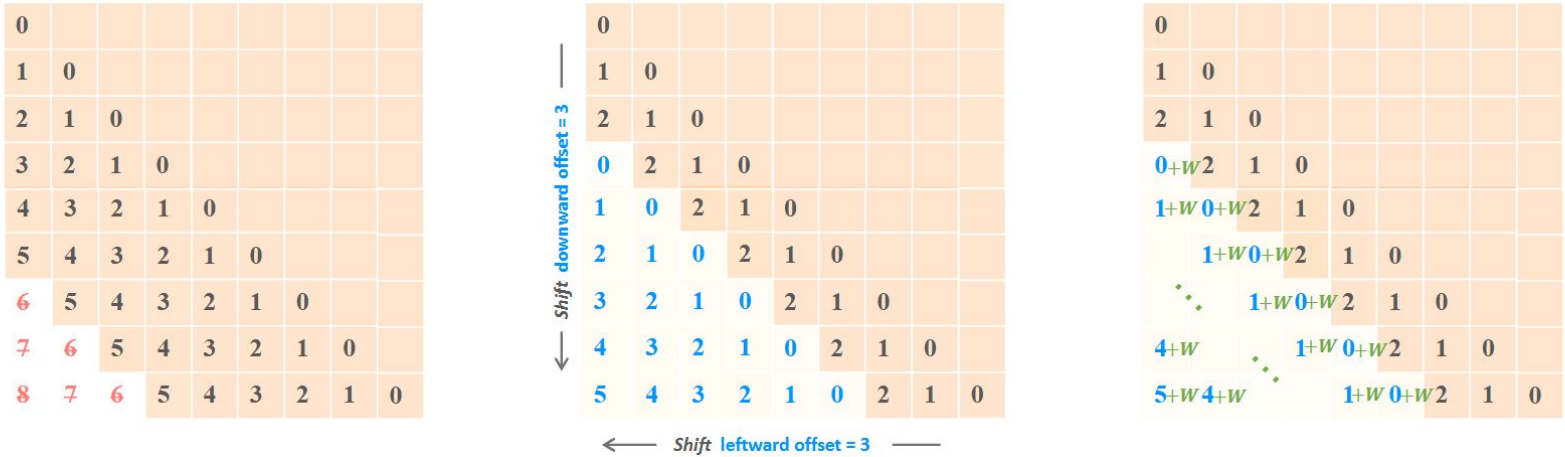
- Model's knowledge on position (i.e. position frequency) is important.
- But model only **knows shorter distance much more than longer ones**.
- Long distance dependency is also **Long Tail** in natural corpus.

Core Idea:

Instead of training with longer positions, let's shift position indices for RoPE, so that long-range indices are now under well-trained positions.

STRING:

Why after messing the up positions, we would still it expect to work?



(a) Dropping infrequent positions (b) Shifting frequent positions (c) Recovering locality

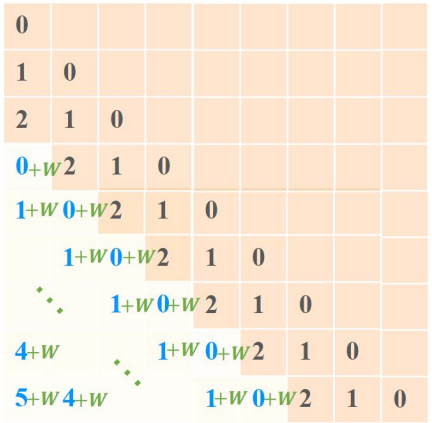
Figure 5: A illustrative example of STRING for a sequence length of $L = 9$. (a) Position indices 6, 7, and 8 are removed from the matrix. (b) Indices 0, 1, 2, 3, 4, and 5 are shifted from the main diagonal to the lower-left triangle with an offset of 3. (c) A small constant W is added to all diagonals where $m \geq n - 3$, thereby restoring emphasis on the neighboring W tokens. The position matrix of Llama3.1-128K using STRING is shown in Figure 8 Appendix.

STRING: Simple Training-fRee embeddING

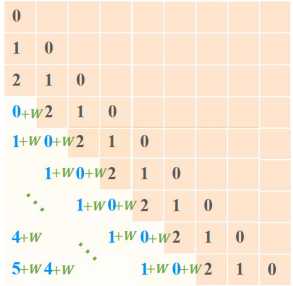
$$P[m][n] = \begin{cases} P[m][n] - S + W & \text{if } m \geq n - S, \\ P[m][n] & \text{otherwise.} \end{cases} \tag{4}$$

Algorithm 1 Pseudocode of STRING with FlashAttention

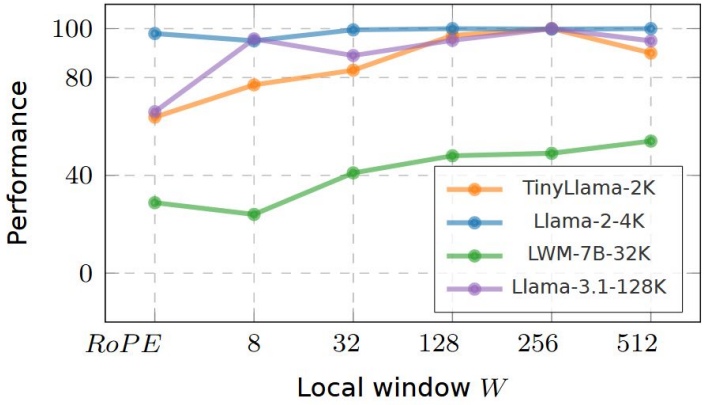
```
# Q, K, V: tensors with shape [L, d]
# W:  the local window value (scalar)
# S:  the slding window size (scalar)
# N:  the left-bottom triangle height (scalar)

pids_query = [0,1,2,...L-1] # standard position ids for keys
pids_key = [0,1,2,...L-1] # standard position ids for queries
# Apply rotary position embeddings to K
K = apply_rotary_pos_emb(K, pids_key)

# <--- Calculating sliding window attention around the diagonal --->
Q_diag = apply_rotary_pos_emb(Q, pids_query)
O_diag, attn_map_diag = flash_attn(Q_diag, K, V, sliding window=S)

# <--- Calculating self-attention at the left-bottom triangle --->
pids_q_shifted = pids_query - S + W # new position ids for queries
Q_shifted = apply_rotary_pos_emb(Q, pids_q_shifted)
# obtain q,k,v in the bottom-left corner & calculate flash-attn
O_shifted, attn_map_shifted = flash_attn(Q_shifted[-N:], K[:N], V[:N])

# Merge the attention outputs from the diagonal and left-bottom triangle
output = merge_diag_shifted(O_diag, O_shifted, attn_map_diag, attn_map_shifted)
```



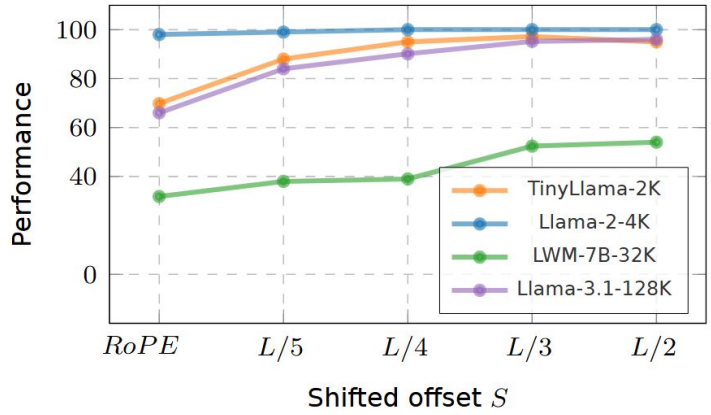
STRING: Parameters



- W from 4 to 512
  - When W ≥ 32, performance improved significantly compared to RoPE
  - As long as W ≪ S, further increasing W does not cause a performance drop.
- S from L/5 to L/2 .
  - As S increases, more position indices are discarded, the performance increases.
  - The trend slowed down when S > L/3, indicating that at least the last 33% to 50% of the position can be overwritten.



(a) Ablation on local window W ($S = \frac{L}{3}$)



(b) Ablation on shifted offset S ($W = 128$)

Experimental Setup: Baselines

- **NTK-Aware RoPE**: Adjusts the RoPE base frequency to scale rotational wavelengths for longer contexts.
- **YaRN**: Introduces temperature scaling to the attention softmax to stabilize score distributions.
- **Resonance RoPE**: Clamps the query's position index to a known maximum for long-range interactions.
- **Self-Extend**: Remaps the key's position index based on its distance from the query during inference.
- **DCA**: Adds a cross-attention mechanism for each layer to query a summary of all preceding layers.

Experimental Setup

- **STRING** focus on improving the performance within the training context size.
- **NTK-Aware RoPE** and **YaRN** implement extrapolation by increasing the base frequency of RoPE.
- Meanwhile, **ReRoPE**, **Self-Extend**, and **DCA** modify the position matrix to avoid unseen positions.
- The training length of the model to 2/3 of the original length and set the extrapolation scaling factor to 3/2, meaning the test sequence length is 1.5 times the training length.

Main Results

Table 1: Needle-in-a-haystack (4 needles) results of 7 base models across various methods (columns reordered from smallest to largest average) where L_{train} means the size of the training context window. All the models were tested using their training length. The number of test cases is 500.

Model	L_{train}	ReRoPE	NTK	RoPE(origin)	Self-Extend	YaRN	DCA	STRING
TinyLlama-1.3B (ours)	2k	62.8	62.0	56.6	60.2	68.6	74.4	**84.6**
TinyLlama-1.1B-3T	2k	77.2	79.8	69.8	83.2	88.0	80.2	**97.2**
Llama-2-7B	4k	98.6	98.6	98.0	95.4	98.0	91.6	**100.0**
Llama-3-8B	8k	99.6	100.0	99.8	99.8	100.0	99.9	99.6
LWM-7B-base	32k	25.2	19.4	31.8	29.0	22.2	28.8	**50.4**
Mistral-7B-base	32k	54.5	42.2	52.8	54.2	48.2	64.2	**73.0**
Llama-3.1-8B	128k	53.6	71.2	66.0	65.8	68.8	72.8	**95.2**
Average	–	67.3	67.6	67.8	69.6	70.5	73.1	**85.7**

NO TRAINING!

Main Results

Models	Effective/Claimed	NIAH	VT	Aggregation	QA	Avg. (13 tasks)
Llama2-chat	4K / 4K	96.9	89.7	84.8	49.7	85.6
GPT-4-1106-preview	64K / 128K	84.8	99.6	79.7	59.0	81.2
GLM4 (*Open-source best*)	64K / 1M	94.4	97.7	49.7	63.6	83.1
LWM (7B)	4K / 128K	83.4	15.2	29.1	52.6	65.0
Phi3-medium (14B)	8K / 128K	51.3	26.0	43.5	38.0	46.1
Llama3.1 (8B)	32K / 128K	92.6	70.4	36.2	58.8	77.0
+ YaRN	32K / 128K	94.7	39.8	38.2	58.8	76.3
+ DCA	32K / 128K	89.5	62.5	39.2	55.2	74.4
+ Self-Extend	32K / 128K	**94.9**	65.0	37.3	49.8	76.8
+ ReRoPE	32K / 128K	90.0	56.3	38.7	56.9	74.4
+ STRING	32K / 128K	94.0	88.1	37.6	62.7	80.0
Yi (34B)	32K / 200K	90.2	76.8	43.4	59.9	77.3
GradientAI/Llama3 (70B)	16K / 1M	84.9	56.2	41.4	59.8	72.1
Mixtral (8x22B)	32K / 64K	23.8	0.0	69.7	40.8	31.7
Command-R-plus (104B)	32K / 128K	65.7	97.2	59.5	39.2	63.1
Llama3.1 (70B)	64K / 128K	78.9	59.2	39.8	47.6	66.6
+ STRING	100K / 128K	92.7	95.6	50.0	**63.0**	81.7
Qwen2 (72B)	64K / 128K	48.0	79.0	70.3	47.2	53.7
+ STRING (*new SOTA*)	100K / 128K	91.2	**98.4**	**83.7**	52.2	**84.6**
Test Length: *100K*						
Llama3.1-STRING (70B)	100K / 128K	94.6	97.8	72.1	67.3	87.2
Qwen2-STRING (72B)	100K / 128K	93.9	97.7	88.1	57.8	87.8

Main Results

Table 3: Comparison of STRING with three leading commercial long-context models on InfiniteBench. Each model is evaluated using a maximum context length of 128K.

Tasks	Commercial Models			Llama3.1 8B		Llama3.1 70B	
	GPT-4	Claude2	Kimi-chat	$\text{RoPE}_{(\text{origin})}$	STRING	$\text{RoPE}_{(\text{origin})}$	STRING
En.Sum	14.73	14.45	17.93	26.00	**28.22**	26.89	27.64
En.QA	**22.22**	11.97	16.52	10.05	10.20	13.68	16.73
En.MC	67.25	62.88	72.49	65.50	70.30	76.41	**81.98**
En.Dia	8.50	**46.50**	11.50	20.00	19.50	18.00	30.50
Retr.PassKey	100.00	97.80	98.14	100.00	100.00	100.00	**100.00**
Retr.Number	100.00	98.14	94.42	99.32	99.89	100.00	**100.00**
Retr.KV	**89.00**	65.40	53.60	42.00	83.00	2.22	76.07
Code.debug	**39.59**	2.28	18.02	22.84	26.90	29.20	32.80
Math.find	**60.00**	32.29	12.57	32.18	34.87	40.92	46.28
Avg.	55.69	47.96	43.91	46.43	52.54	45.25	**56.88**

Conclusions

- STRING achieves highest performance on all 7 models in 4-NIAH evaluation (2k-128k lengths)
  - Extrapolation methods fail within training windows
- STRING extends effective context length to 100k (up from 64k)
- Improvements hold from controlled tasks to practical applications
- Benefits increase with model size
- Results suggest that addressing undertrained positions improves long-context performance

Critiques

- Is it fair to use extrapolation methods as a comparison? (⅔ compression * 3/2 extension)
- STRING results are impressive, but do they create position ambiguity?
  - well-trained positions become pleiotropic by design → possibly introduce issues with superposition/representation interference?
  - relying on content to disambiguate → does STRING struggle when content is repetitive?
- Claim: position frequency during training → quality of learned attention patterns for that relative distance
  - High position frequency → better learned attention → better retrieval and context utilization
  - if reusing positions works, why didn't training learn to make all positions work? positions are just indices in the same RoPE encoding function
- STRING *fits* model bias, doesn't correct it, fundamental architectural constraints remain
  - model's just need *sufficient* frequency for positions and content does the rest

Future Experiments

- Evaluation is biased towards retrieval, let’s see some long-form generation
- When does STRING fail?
- STRING leaves a gap in position indices, why doesn’t this matter?
  - Pretty crazy that losing ~⅓ of position indices doesn’t hurt the model
- Some attention visualization could help explain *why* STRING works so well
  - Show how the model effectively reuses positions
- Is STRING effective for SFT?
- Authors should train with forced uniform position distribution
  - Train with oversampled long sequences, or gradient reweighting 1/f(i)
  - If this fixes the problem, then position frequency is causal
- Test position-critical tasks (find the Nth occurrence of X)
  - should test limits of content disambiguation

Further Readings

1. Dissecting Transformer Length Extrapolation via The Lens of Receptive Field Analysis
2. Information Entropy Invariance: Enhancing Length Extrapolation in Attention Mechanisms

Appendix

The Foundation: Rotary Position Embedding (RoPE)

Concept and Mathematics

The core idea is to encode absolute position using rotation matrices, such that the dot product in self-attention naturally captures relative positional information.
A vector $v \in \mathbb{R}^d$ is viewed as $d/2$ 2D vectors, each rotated at position m by a matrix R_m:

$$R_m = \begin{pmatrix} \cos m\theta_0 & -\sin m\theta_0 & \cdots & 0 \\ \sin m\theta_0 & \cos m\theta_0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \cos m\theta_{d/2-1} \end{pmatrix}$$

where the frequencies are $\theta_i = b^{-2i/d}$. The transformed query $q'_m = R_m q$ and key $k'_n = R_n k$ have a dot product that depends only on the relative position $m - n$:

$$(q'_m)^T k'_n = (R_m q)^T (R_n k) = q^T R_m^T R_n k = \mathbf{q^T R_{n-m} k}$$

1. NTK-Aware RoPE

To handle sequences longer than the training length (L_{train}), NTK-Aware RoPE scales the base b of its frequencies. This "stretches" the rotational wavelengths, effectively turning extrapolation into interpolation.

The Math

Define the scaling factor $\lambda = L_{extend}/L_{train}$. The base b is adjusted to b':

$$b' = b \cdot \lambda^{d/(d-2)}$$

This changes the frequencies θ_i to θ'_i, reducing the rotation angles:

$$\theta'_i = (b')^{-2i/d} = \theta_i \cdot \lambda^{-2i/(d-2)}$$

2. YaRN: Yet another RoPE extensioN

YaRN improves upon NTK-scaling by using a more nuanced interpolation and introducing a temperature scaling factor to the attention mechanism, preserving the attention score distribution for longer contexts.

The Math

The main addition is the temperature t in the softmax. Given scale factor $s = L_{extend}/L_{train}$:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\mathbf{t}\sqrt{d_k}}\right)V$$

The temperature t is an empirically derived function of the scale factor s:

$$t = \left(\frac{s \log L_{train}}{\log(sL_{train})}\right)^{0.25}$$

3. Resonance RoPE (ReRoPE)

When an out-of-distribution (OOD) query attends to a very distant key, it creates an unseen relative distance. ReRoPE solves this by "clamping" the query's position index for long-range attention, ensuring the rotation is within a familiar range.

The Math

Let L be the training context length. For a query q_m and key k_n:

$$\text{AttentionScore} = \begin{cases} (R_m q_m)^T (R_n k_n) & \text{if } m - n < L \\ (R_{L-1} q_m)^T (R_n k_n) & \text{if } m - n \geq L \end{cases}$$

4. Self-Extend: Tuning-Free Extension

For tuning-free context extension at inference time, Self-Extend remaps the position indices passed to RoPE. This ensures that all computed relative distances fall within the range the model was trained on.

The Math

Let L be the training length and $W = L/2$ be the neighbor window. For a query at position i attending to a key at j, the key's position is re-mapped to j':

$$j' = \begin{cases} j - (i - W) & \text{if } i - W < j \leq i \quad \text{(Neighbor)} \\ j \pmod{W} & \text{if } j \leq i - W \quad \text{(Grouped/Distant)} \end{cases}$$

5. DCA: DeepCrossAttention (Architectural Improvement)

DCA is not a context extension method, but an architectural improvement. It enhances the residual stream by propagating a "summary" state C_l through layers, which each subsequent layer can query via cross-attention.

The Math

At each layer l, the summary context C_l is updated from the hidden state H_l:

$$C_{l+1} = (1 - \alpha_l)H_l + \alpha_l C_l \quad (\alpha_l \text{ is a learnable gate})$$

Layer $l + 1$ uses cross-attention where Q is from H_l and K, V are from C_{l+1}:

$$Q_{l+1} = H_l W_Q \quad | \quad K_{l+1} = C_{l+1} W_K \quad | \quad V_{l+1} = C_{l+1} W_V$$

The layer's output updates the residual stream: $H_{l+1} = H_l + \text{DCA}(H_l, C_{l+1})$.