Pre-RoPE versus Post-RoPE: Trading Per-Step Compute for KV-Cache Survivability

Gnosis Research Center @ ILLINOIS TECH

Zia Uddin Chowdhury Jaime Cernuda Jie Ye Anthony Kougkas
Xian-He Sun
{zchowdhury1, jye20}@hawk.illinoistech.edu, {jcernudagarcia, akougkas, sun}@illinoistech.edu

Motivation

- LLM inference engines cache K, V to avoid recomputing past representations every decoding step.
- **Post-RoPE storage** (today's default) bakes position into every cached key: identical content at different positions cannot share a cache entry, bloating the KV-cache footprint in agentic, multi-turn, and RAG workloads.
- Recent systems already lean on the **pre-RoPE** alternative: KVQuant for cleaner per-channel quantization, CacheBlend for cross-prompt KV reuse via delta-rotation realignment, and CachedAttention for decoupled positional encoding to keep caches valid under truncation.

Background

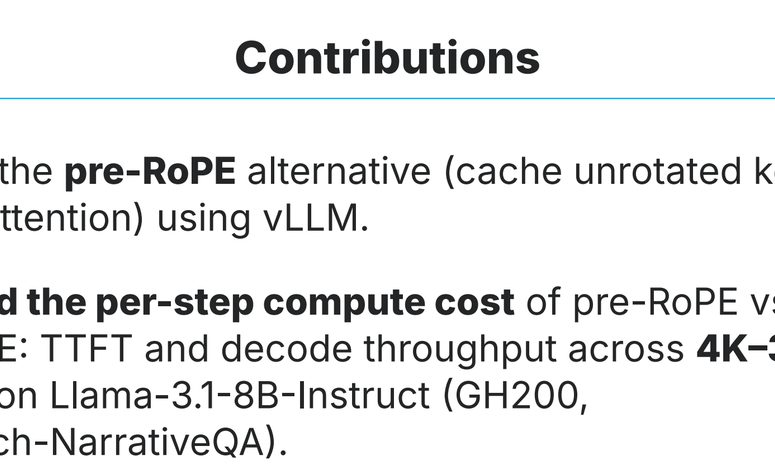
- **Prefill:** processes the entire prompt in one pass and emits the first token. Latency = **TTFT**.
- **Decode:** generates one token per step, attending over the whole cache. Steady-state speed = **throughput**.
- **RoPE:** rotates each Q, K by an angle tied to absolute position; attention only sees the *relative* distance $i - j$.
- **Truncation & Perplexity (PPL):** simulates removing the earliest tokens from the cache when context exceeds capacity. **PPL** is a metric for the quality of token generation (*lower = better next-token prediction*).

Contributions

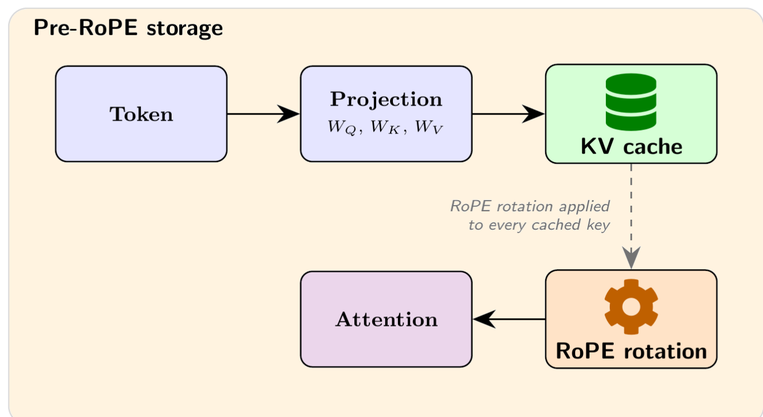
- Modeled the **pre-RoPE** alternative (cache unrotated keys; apply RoPE at attention) using vLLM.
- **Measured the per-step compute cost** of pre-RoPE vs post-RoPE: TTFT and decode throughput across **4K–32K** contexts on Llama-3.1-8B-Instruct (GH200, LongBench-NarrativeQA).
- **Measured the truncation survivability benefit**: under 50% cache truncation, pre-RoPE re-rotation is $\sim$**30**$\times$ **faster** than the re-prefill baseline and yields $\sim$**2**$\times$ **lower perplexity** than naive post-RoPE slicing.

Implementation

Post-RoPE fuses the rotation into the K projection: rotated keys are written into the cache and read directly at attention, so no per-step rotation overhead.



Pre-RoPE writes unrotated keys; before every forward pass, all cached keys are re-rotated to their current positions, attention runs, and the unrotated originals stay in the cache. Cost: $O(N)$ per layer per decode step.

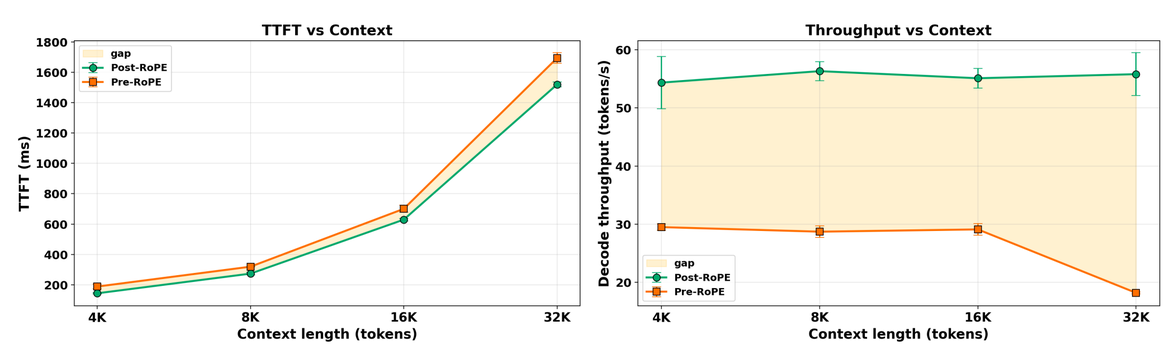


Evaluation Setup

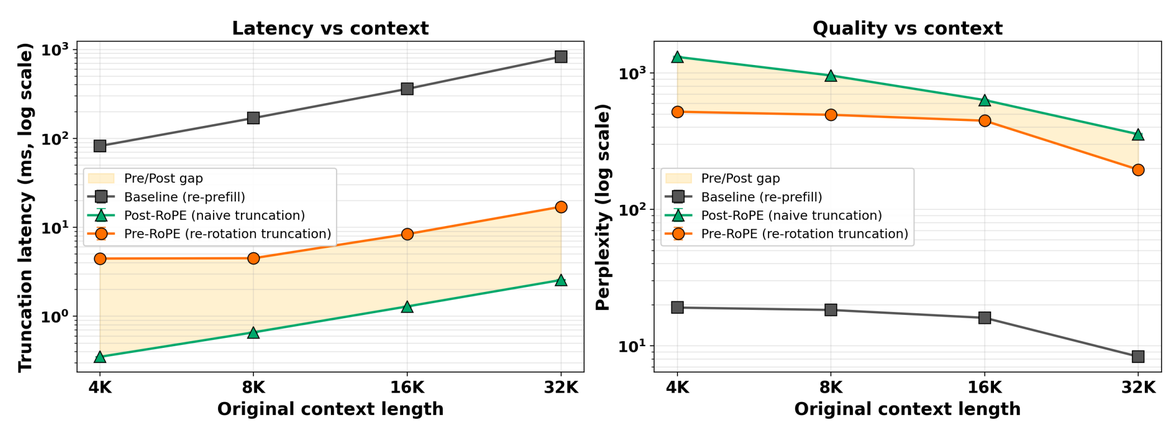
Model	Llama-3.1-8B-Instruct
Engine	vLLM
Hardware	NVIDIA GH200 (Delta)
Dataset	LongBench (NarrativeQA)
Contexts	4K / 8K / 16K / 32K
Metrics	TTFT, decode throughput, perplexity

Results

We measure both sides of the trade-off: the steady-state per-step compute cost of pre-RoPE storage, and the cache survivability it enables when context overflows and the cache has to be truncated.



Baseline / Post-RoPE / Pre-RoPE across context lengths (for 50% truncation)



- **Cost.** Pre-RoPE TTFT runs 10–30% higher than post-RoPE across context lengths, and decode throughput is 1.8–3.1x lower.
- **In return**, when 50% of the cache is truncated, pre-RoPE re-rotation completes in $\sim$30x less time than the re-prefill baseline and stays within $\sim$2x of its perplexity.
- **By comparison**, naive post-RoPE slicing inflates perplexity by 30–65x relative to baseline, since surviving keys carry stale RoPE rotations encoding their pre-truncation positions, which distorts subsequent attention.
- **However**, truncation is one such case where pre-RoPE's properties are favoured: the cost recurs throughout generation while the survivability benefit triggers only at truncation events, so deployments with frequent context overflow see the largest net win.

Conclusion

Pre-RoPE trades per-step compute for **KV-cache survivability under truncation**: on overflow, post-RoPE forces a choice between full re-prefill and quality collapse (its rotated K carries stale positions after the slice), while pre-RoPE recovers most of the quality at a fraction of the cost.