Dropping the Anchor: Statistical Context Summarization for Distributed Systems via Pulsar Attention

Anonymous ACL submission

Abstract

Inference with large language models (LLMs) on long sequences is computationally expensive due to the quadratic complexity of self-attention. Distributed blockwise methods such as Star Attention reduce this cost by sharding context across hosts, but rely on prepending a static, content-blind copy of the first block to every host. We propose Pulsar Attention, which replaces the static anchor with two lightweight, content-aware components: a small attention-sink prefix that stabilizes softmax, and compact cross-block summaries built via a Max-IDF heuristic that selects chunks containing globally rare tokens. This reduces the Phase 1 per-GPU FLOPs by up to 3.3× over Star Attention while retaining an identical KV cache footprint. On RULER with Llama-3.1-8B-Instruct, Pulsar Attention outperforms both Star Attention and dense attention at sequence lengths up to 128K tokens, with absolute gains of up to 4.7% over the dense baseline.

1 Introduction

Large Language Models (LLMs) have demonstrated strong performance on tasks requiring long context windows, including repository-level code generation, multi-document summarisation, and long-horizon retrieval. However, scaling inference to these lengths remains expensive: self-attention is quadratic in sequence length, and the Key–Value (KV) cache that avoids redundant computation during autoregressive decoding grows linearly, quickly exhausting device memory on a single GPU.

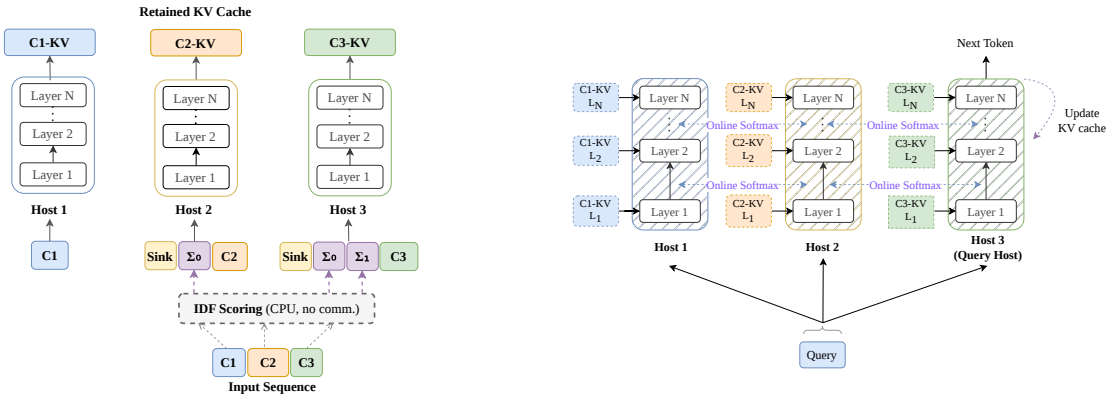
Distributed inference methods address this bottleneck by sharding context across multiple GPUs. Ring Attention (Liu et al., 2023) computes exact global attention by circulating KV blocks in a ring topology, but pays for correctness with coordinated cross-host communication at every layer. Star Attention (Acharya et al., 2025) eliminates this communication during context encoding by splitting the sequence into independent blocks processed in parallel, then recovering global attention at query time through a lightweight softmax merge. The key enabler is an anchor block: a copy of the first block prepended to every host's input. This stabilizes the blockwise attention distribution. However, the anchor duplicates the same tokens regardless of their relevance, doubling per-host sequence length during context encoding and providing no information from intermediate blocks (Section 2).

We propose Pulsar Attention, which replaces the static anchor with two lightweight, content-aware mechanisms: (i) a small attention-sink prefix (64 tokens from the sequence start) that stabilizes the softmax distribution at a fraction of the anchor's cost, and (ii) statistical block summaries, constructed by scoring contiguous chunks with a Max-IDF heuristic and retaining only the top-k most informative chunks per block, which propagate causally from earlier blocks to later ones. Unlike the anchor, these summaries adapt to each block's content, prioritizing chunks containing tokens most likely to carry task-critical information such as entity names, identifiers, or numerical values.

The result is a strict reduction in Phase 1 compute: each host processes its local block plus a small, fixed-size prefix rather than a full duplicate block, reducing per-GPU FLOPs by up to 3.3× relative to Star Attention while retaining an identical KV cache footprint. On RULER, Pulsar Attention outperforms both Star Attention and dense attention from 32K to 128K tokens, with absolute accuracy gains of up to 4.7% over the dense baseline, suggesting that content-aware summarization not only saves compute but actively improves attention quality by filtering out dilutive distractor tokens.

Our main contributions are:

- We replace Star Attention's static anchor with content-aware summaries and a small attention sink, reducing Phase 1 attention FLOPs



(a) Phase 1: Statistical context encoding. Host i assembles input $[\mathcal{S}_{\text{sink}} \parallel \Sigma_0 \parallel \cdots \parallel \Sigma_{i-1} \parallel B_i]$, runs a forward pass, then discards sink and summary KV entries, retaining only $\text{KV}(B_i)$.

(b) Phase 2: Global query encoding. Each host attends over its retained KV shard; an `all_gather` and online softmax merge (Milakov and Gimelshein, 2018) recovers exact global attention without transmitting the full KV cache.

Figure 1: Pulsar Attention overview. *(Left)* Phase 1 replaces the static anchor block with a content-aware prefix: a small attention-sink and Max-IDF statistical summaries of all causally preceding blocks. *(Right)* Phase 2 broadcasts the query to all hosts and merges local attention scores into an exact global output via online softmax.

by up to $3.3\times$ while retaining an identical KV cache footprint.

- We introduce a Max-IDF scoring mechanism and validate it against three alternative heuristics on BABILONG across multiple context lengths, showing that rare-token-based selection effectively captures cross-block context.
- We demonstrate empirically on RULER that Pulsar Attention outperforms both Star Attention and dense attention at context lengths up to 128K, with gains of up to $+4.7\%$ over the dense baseline, and demonstrate competitive accuracy on Qwen3-4B-Instruct without architecture-specific tuning, confirming cross-architecture applicability.
- We validate FLOPs-based efficiency estimates with end-to-end wall-clock profiling and GPU memory measurements, confirming speedups of up to $6.24\times$ over dense attention at 128K.

2 Background

In standard Transformer-based LLMs, autoregressive decoding relies on self-attention computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V \quad (1)$$

where $Q, K, V \in \mathbb{R}^{L\times d}$ are the query, key, and value matrices for a sequence of length L and head dimension d. To avoid redundant recomputation, LLMs cache the key and value vectors of all previous tokens during generation. While this optimizes compute, the KV cache grows linearly with sequence length and the attention computation remains quadratic in L, quickly exhausting device memory and wall-clock budget at long contexts.

Liu et al. (2023) addresses this by distributing the KV cache across GPUs and circulating blocks in a ring to compute exact global attention. This enables arbitrarily long sequences but requires cross-host communication at every layer, introducing synchronization overhead that scales with model depth.

Acharya et al. (2025) eliminates this communication during context encoding with a two-phase design. In Phase 1, the context is split into contiguous blocks processed independently on parallel hosts. Without any shared context, blockwise attention produces spurious per-block attention sinks; tokens at the start of each block accumulate disproportionate softmax mass simply by virtue of their local position, creating a degenerate attention distribution that fails to approximate global attention. To prevent this, a copy of the first block (the anchor block) is prepended to each host's input. The anchor serves a dual role: its initial tokens absorb the attention sink effect (Xiao et al., 2023), redirecting spurious sinks away from block-internal positions, while its semantic content provides cross-block context that later blocks need for accurate attention during Phase 2. Ablation experiments in

Acharya et al. (2025) confirm that anchor content is critical; replacing it with constant or random tokens causes accuracy degradation; while the anchor's positional placement has comparatively minor effect. In Phase 2, the query is broadcast to all hosts, which compute local attention scores and aggregate them via an online softmax merge (Milakov and Gimelshein, 2018) to recover global attention.

While effective, the anchor is content-fixed: it always duplicates the first block regardless of its relevance to any given later block, doubling per-host sequence length and FLOPs during Phase 1. A block i receives no information about intermediate blocks 2 through $i-1$, and as sequences grow longer the gap between anchor and local block widens, meaning this static duplication captures a diminishing fraction of the cross-block context. Our method addresses this limitation.

3 Methodology

3.1 Phase 1: Statistical Context Encoding

Like Star Attention, Pulsar Attention partitions the input sequence of length L into B contiguous blocks $\{B_0, B_1, \ldots, B_{n-1}\}$, each assigned to a parallel host (Figure 1a). Rather than duplicating the entire first block as a static anchor, we replace it with two lightweight, content-aware components: an attention-sink prefix and Max-IDF block summaries. Context encoding proceeds in four steps.

Attention sinks. The first s_{sink} tokens of B_0 are extracted as a fixed prefix $\mathcal{S}_{\text{sink}}$ (default $s_{\text{sink}} = 64$). Following Xiao et al. (2023), these tokens stabilize the softmax distribution during blockwise attention at a fraction of the cost of a full anchor block.

Summary generation. Before any neural computation, every host constructs an identical corpus-level IDF table from the token IDs of the full sequence, an $O(L)$ integer-only operation that requires no inter-host communication and no GPU time. Using this table, each host scores non-overlapping contiguous chunks of m tokens (default $m = 32$) within each block via Max-IDF:

$$\text{score}(C) = \max_{t \in C} \text{IDF}(t) \tag{2}$$

$$\text{IDF}(t) = \log\left(\frac{N}{\max(\text{df}(t), 1)}\right) \tag{3}$$

where $\text{df}(t)$ is the number of blocks containing token t. The top-k chunks per block are selected and concatenated in positional order to form block summary Σ_i, with a per-block budget of σ tokens

Algorithm 1 Pulsar Attention

Input: Blocks $\{B_0, \ldots, B_{n-1}\}$, query Q, sink size s_{sink}, chunk size m, chunks per block k, max tokens T
Output: Generated token sequence
$\mathcal{S}_{\text{sink}} \leftarrow B_0[0{:}s_{sink}]$; build $\text{IDF}(t)$ from $\text{df}(t)$ across all blocks
{*Phase 1: Statistical Context Encoding*}
for host $i = 0, \ldots, n-1$ **in parallel do**
  $\Sigma_i \leftarrow$ top-k chunks of B_i by $\max_{t \in C} \text{IDF}(t)$, in positional order
  $\text{Input}_i \leftarrow [B_0]$ if i=0, else $[\mathcal{S}_{\text{sink}} \,\|\, \Sigma_0 \,\|\, \cdots \,\|\, \Sigma_{i-1} \,\|\, B_i]$
  $\text{KV}_i \leftarrow$ FORWARDPASS(Input_i); discard non-B_i KV entries
end for
{*Phase 2: Query Encoding & Generation*}
Broadcast Q to all hosts; $q \leftarrow Q$; output $\leftarrow [\,]$
for $t = 1$ **to** T **do**
  $(A_h, \ell_h) \leftarrow$ FLASHATTN(q, KV_h) for each host h in parallel; `all_gather` $\{(A_h, \ell_h)\}$
  $(A, \ell) \leftarrow$ ONLINESOFTMAXMERGE$\big(\{(A_h, \ell_h)\}_{h=0}^{n-1}\big)$
  $q \leftarrow \text{argmax}(\text{LMHEAD}(A))$; append $\text{KV}(q)$ to KV_{n-1}; append q to output; **if** $q =$ `EOS` **then break**
end for
return output

(default $\sigma = 0.125 \times |B_i|$). We choose contiguous chunks rather than individual tokens to preserve local syntactic structure in the resulting key–value representations. Max-IDF is preferred over averaging heuristics because identifiers, entities, and numerical values often appear as isolated high-IDF tokens that dominate retrieval relevance in long-context QA because averaging dilutes this signal by a factor of $1/m$ (Section 5.1).

Causal Assembly. Each host i prepends only summaries from causally preceding blocks along with $\mathcal{S}_{\text{sink}}$, forming the augmented Phase 1 input:

$$\text{Input}_i = \begin{cases} [B_0] & i = 0 \\ [\mathcal{S}_{\text{sink}} \| \Sigma_0 \| \cdots \| \Sigma_{i-1} \| B_i] & i > 0 \end{cases} \tag{4}$$

Each token retains its original global position ID, preserving the inter-token distances encoded by RoPE (see Appendix E for a comparison with contiguous re-numbering).

KV Discard. After the forward pass, KV states

Table 1: RULER average accuracy (%) on Llama-3.1-8B-Instruct. †From Acharya et al. (2025), same model and setup. $B = 4$ for 16K-64K; $B = 8$ for 128K. **Bold** = best among efficiency-focused methods.

Method	Setting	16K	32K	64K	128K	Avg
Full Attention	Single GPU	92.2	87.5	84.8	76.3	85.2
StreamingLLM†	Single GPU	74.8	48.6	26.2	30.8	45.1
MInference†	Single GPU	93.3	86.5	84.9	58.2	80.7
Star Attention†	Multi-GPU	91.3	88.7	83.4	74.4	84.4
Pulsar Attention (Ours)	Multi-GPU	90.4	**91.5**	**89.5**	**79.1**	**87.6**

for $\mathcal{S}_{\text{sink}}$ and $\{\Sigma_j\}_{j<i}$ are discarded; only the $|B_i|$ entries belonging to the block itself are retained. The final distributed KV cache spans exactly L tokens with no duplication, and peak per-host memory during Phase 1 is strictly lower than Star Attention's: each host processes at most $L/n + s_{\text{sink}} + C$ tokens versus Star's $2L/n$, where $C \ll L/n$ is the total summary budget.

3.2 Phase 2: Global Query Encoding and Generation

Because Phase 1 produces a clean, non-overlapping distributed KV cache, Phase 2 is structurally identical to Star Attention (Acharya et al., 2025): the query is broadcast to all hosts, each computes local attention via FlashAttention, and a designated query host aggregates the results through an online softmax merge (Milakov and Gimelshein, 2018). The full pseudocode is given in Algorithm 1.

4 Experiments

4.1 Experimental Setup

Models. Our primary evaluation uses Meta-Llama-3.1-8B-Instruct (Grattafiori et al., 2024), which supports a native 128K context window. To assess cross-architecture generality, we also evaluate on Qwen3-4B-Instruct (Qwen Team, 2025). All experiments use bf16 precision with FlashAttention-2. Generation uses greedy decoding.

Distributed configuration. Following Star Attention, the context is partitioned into $B = 4$ equal blocks, with one block assigned per GPU. All distributed experiments use 4 hosts for all context lengths. However, we switched to 8 blocks, i.e., two blocks processed per host for 128K due to memory efficiency. The summary budget is fixed at 12.5% of block size (chunk size m=32 tokens, with the number of selected chunks k scaled proportionally to context length). The attention-sink prefix is fixed at 64 tokens across all settings.

Benchmarks. We evaluate on two benchmarks: (i) RULER (Hsieh et al., 2024), a synthetic long-context benchmark comprising 13 tasks across five categories (single-needle retrieval, multi-needle retrieval, multi-hop tracing, aggregation, and question answering), with 500 samples per task per context length; and (ii) BABILong (Kuratov et al., 2024), a reasoning benchmark requiring multi-step inference over supporting facts embedded in long distractor text, with 1,000 samples per task. We report exact-match accuracy for both benchmarks.

Baselines. We compare against four baselines: (i) Full Attention, computed on a single GPU as the accuracy ceiling; (ii) Star Attention (Acharya et al., 2025), the most directly comparable distributed method; (iii) StreamingLLM (Xiao et al., 2023), a sparse method combining sink tokens with sliding-window attention; and (iv) MInference (Jiang et al., 2024a), which dynamically selects per-head sparse attention patterns. Baselines marked with † are reproduced from Acharya et al. (2025) using the same model and hardware configuration.

4.2 Results on RULER

Table 1 reports average accuracy across all 13 RULER tasks at context lengths from 16K to 128K. Pulsar Attention outperforms Star Attention at 32K (+2.8%), 64K (+6.1%), and 128K (+4.7%), with the gap widening at longer contexts. At 16K, Pulsar trails Star Attention by 0.9%, likely because the summary budget (512 tokens from a 4K block) provides less coverage than Star's full 4K anchor at short contexts where the anchor's cost is relatively low. This is consistent with the results in Section 5.1: at small budgets ($s = 128$), even the best heuristics trail Star Attention on `qa1`, and only exceed it as the budget grows relative to block size. At 16K the block size is small enough that the anchor's cost is negligible while its coverage benefit remains, narrowing Pulsar's advantage.

Notably, Pulsar also exceeds Full Attention ac-

Table 2: RULER average accuracy (%) on Qwen3-4B-Instruct. **Bold** = best per context length.

Method	16K	32K	64K	128K
Full Attention	**92.4**	89.3	88.6	**86.5**
Pulsar Attention (Ours)	91.6	**90.4**	**88.8**	85.6

curacy at 32K (+4.0%), 64K (+4.7%), and 128K (+2.8%). We hypothesize that restricting Phase 1 attention to local blocks while propagating content-aware summaries may reduce interference from irrelevant tokens at long contexts, yielding sharper softmax distributions than full-sequence attention.

Among the single-GPU baselines, MInference performs competitively at 16K-64K but degrades sharply to 58.2% at 128K (-20.9 points versus Pulsar), suggesting that its offline sparse pattern estimation becomes unreliable at extreme lengths. StreamingLLM collapses past 16K, confirming that recency-biased context windows cannot substitute for cross-block information flow.

To verify cross-architecture generality, we also evaluate on Qwen3-4B-Instruct using the same distributed configuration ($B = 4$, Max-IDF summaries at 12.5% budget; Table 2). Pulsar outperforms Dense at 32K (+1.1%) and 64K (+0.2%), consistent with the Llama-3.1-8B-Instruct pattern, and trails Dense by only 0.8% at 16K and 0.9% at 128K. The latter gap can be explained by Qwen3-4B's higher absolute Dense accuracy (86.5% vs. 76.3%), which leaves less margin for summary-based compensation.

4.3 Results on BABILong

Table 4 reports accuracy on three BABILong tasks (qa1, qa3, qa5) at 16K context length using $B = 4$ blocks of 4K tokens each. Pulsar Attention with Max-IDF scoring ($\sigma = 512$) achieves 37% on qa1, 44% on qa3, and 63% on qa5. On the multi-hop tasks (qa3 and qa5), Pulsar outperforms Star Attention (33% and 58% respectively) by +11% and +5%. The advantage on qa3 is particularly notable: this task requires chaining three supporting facts that may reside in different blocks, and Max-IDF's ability to select chunks containing rare entity tokens provides cross-block context that Star's position-fixed anchor cannot.

4.4 RULER Task-Category Analysis

Table 3 reports per-task-category accuracy at 128K tokens on RULER, where the differences between methods are most pronounced.

Table 3: Per-task accuracy (%) at 128K on Llama-3.1-8B-Instruct ($B = 8$, block size 16K, summary budget 12.5%). NIAH MultiKey and NIAH Single scores are averaged across their three sub-tasks; QA averages SQuAD and HotpotQA.

Task	Dense	Star	Pulsar
NIAH Single (avg)	99.7	99.9	**100.0**
NIAH MultiKey (avg)	**83.9**	80.1	78.7
NIAH MultiValue	**91.6**	82.7	63.5
NIAH MultiQuery	**97.8**	96.6	97.0
Variable Tracking	61.8	**62.8**	61.0
Common Words	0.04	0.04	**81.2**
Frequent Words	72.3	75.9	**80.7**
QA (avg)	**58.8**	54.8	54.5
Overall Avg	76.3	74.4	**79.1**

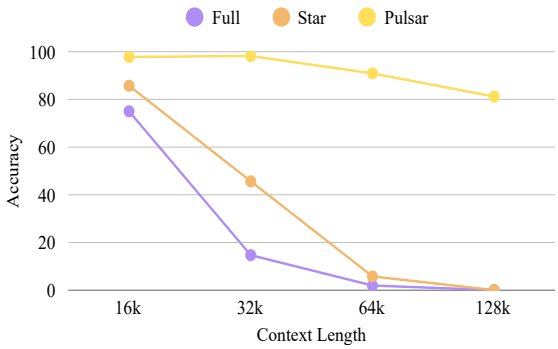


Figure 2: Accuracy on the Common Words Extraction task (RULER) across context lengths using Llama-3.1-8B-Instruct. Full Attention and Star Attention both collapse to near-zero accuracy at 128K, while Pulsar Attention maintains 81.2% by propagating IDF-scored summaries that preserve cross-block frequency.

Retrieval (NIAH). Single-needle retrieval is effectively saturated (100%) across all methods. On MultiKey and MultiQuery, Pulsar maintains near-parity with Dense, as Max-IDF reliably promotes chunks containing rare UUID or keyword needles. The notable exception is NIAH MultiValue (63.5% vs. Dense 91.6%): this task requires retrieving multiple values associated with a single key, where the values may reside in different blocks. Max-IDF selects the chunk containing the rare key identifier but may miss value chunks whose tokens are individually less distinctive.

Aggregation. The most striking result is Common Words Extraction, where both Full Attention (0.04%) and Star Attention (0.04%) effectively fail at 128K, while Pulsar achieves 81.2%. This task requires identifying tokens that appear frequently

Table 4: BABILONG-16K accuracy (%) across three summary budgets(σ). **Bold**: best among the four IDF/entropy-based heuristics (TF-IDF, BM25, Entropy, Max-IDF) per budget–task group. Even-Spaced and Star Attention are shown as non-statistical baselines; Star uses a fixed size anchor block ($B = 4$, block size 4K).

	$\sigma = 128$			$\sigma = 512$			$\sigma = 1{,}024$		
Heuristic	qa1	qa3	qa5	qa1	qa3	qa5	qa1	qa3	qa5
TF-IDF	**40**	33	61	37	30	57	33	30	**60**
BM25	38	34	61	**39**	32	57	33	**35**	**60**
Entropy	36	**39**	65	38	39	62	35	27	**60**
Max-IDF *(ours)*	39	36	**66**	37	**44**	**63**	**36**	34	57
Even-Spaced	49	36	57	44	26	55	46	25	52
Star Attn	35	33	58	35	33	58	35	33	58

across the full sequence, precisely the global co-occurrence signal that dense attention loses at long contexts and that a position-fixed anchor cannot reconstruct. Pulsar's causal summaries propagate recurring tokens from earlier blocks into each host's local context, restoring partial global frequency information without transmitting the full KV cache. Frequent Words Extraction shows the same pattern at a smaller magnitude (Pulsar 80.7% vs. Dense 72.3% vs. Star 75.9%).

Multi-hop reasoning. Variable Tracking and QA show broad parity across all three methods (within 4 points), consistent with the expectation that multi-hop inference benefits from cross-block context but ultimately requires denser inter-block communication than the current summary budget provides.

5 Ablations

5.1 Scoring Heuristic Comparison

Table 4 compares Max-IDF against three alternative scoring heuristics (TF-IDF, BM25, Entropy) and two non-statistical baselines (Even-Spaced sampling and Star Attention's fixed anchor) on BABILong-16K at three summary budgets ($\sigma \in \{128, 512, 1024\}$). Extended results at 32K and 64K are reported in Appendix G.

At the $\sigma = 512$ budget, **MAX-IDF** achieves the best multi-hop accuracy: qa3 rises to 44% (+11% over Star Attention) and qa5 reaches 63% (+5% over Star Attention), establishing clear margins on the most discriminative tasks.

Why Max-IDF excels at multi-hop tasks. Tasks qa3 and qa5 require joining multiple evidence spans, each mentioning a specific entity. These entity tokens receive the highest IDF values, yielding $\mathrm{IDF}(t) \approx \log N$. Max-IDF guarantees that any chunk containing a rare entity is selected regardless of surrounding distractor tokens. In contrast, TF-IDF and BM25 *average* IDF over the chunk, so a single high-IDF entity among 31 filler tokens contributes only $\sim 3\%$ of the score, while Entropy is blind to global rarity entirely.

Budget sensitivity and selection. Max-IDF's advantage is not monotonic across all budgets. At $\sigma = 1024$, qa3 drops from 44 to 34 and qa5 from 63 to 57, both below the $\sigma = 512$ level. We attribute this to summary saturation: at large budgets, Max-IDF begins selecting chunks with only moderately rare tokens, introducing distractor content that dilutes prefix quality. At $\sigma = 128$, all heuristics perform comparably, with Entropy slightly leading on qa5 (66% vs. Max-IDF's 65%). These patterns indicate Max-IDF is best suited to moderate budgets (10-15% of block size) where it can remain selective. Based on these trends, we fix $\sigma = 12.5\%$ of block size (512 tokens for a 4K block) in all experiments, large enough to cover multi-hop entity chains and small enough to limit Phase 1 overhead and avoid saturation.

5.2 Computational Efficiency

We derive Phase 1 memory and FLOPs analytically for Llama-3.1-8B-Instruct (Grattafiori et al., 2024) ($L_{\text{layers}} = 32$, $H_Q = 32$, $H_{KV} = 8$, $d_h = 128$, fp16), using $B = 4$, $\sigma = 512$, $s_{\text{sink}} = 64$. Under FlashAttention (Dao et al., 2022), HBM activation memory scales as $O(n)$ while FLOPs remain $\Theta(n^2 d)$.

Critical-path sequence length. The worst-case per-GPU input (last block with all prior summaries prepended) is:

$$\begin{aligned} n_{\text{Dense}} &= L, \\ n_{\text{Star}} &= \tfrac{L}{2}, \\ n_{\text{Pulsar}} &= \tfrac{L}{4} + \underbrace{s_{\text{sink}} + (B-1)\,\sigma}_{=1{,}600}. \end{aligned} \tag{5}$$

Table 5: Phase 1 per-layer computational cost for the critical (last) block on Llama-3.1-8B-Instruct ($B = 4$, $\sigma = 512$, $s_{\text{sink}} = 64$, bf16). All three columns: critical-path sequence length, peak activation memory, and FLOPs are reported per layer for the worst-case host.

	Critical-path length n			Peak act. mem. / layer			Phase 1 FLOPs / layer		
Method	16K	32K	64K	16K	32K	64K	16K	32K	64K
Dense (serial)	16,384	32,768	65,536	320 MB	640 MB	1.25 GB	2,749 G	10,995 G	43,981 G
Star Attn	8,192	16,384	32,768	160 MB	320 MB	640 MB	687 G	2,749 G	10,995 G
Pulsar *(ours)*	5,696	9,792	17,984	117 MB	201 MB	368 MB	332 G	982 G	3,312 G
Reduction vs. Dense									
Star Attn	2.0×	2.0×	2.0×	2.0×	2.0×	2.0×	4.0×	4.0×	4.0×
Pulsar *(ours)*	2.9×	3.4×	3.6×	2.9×	3.4×	3.6×	8.3×	11.2×	13.3×
Pulsar reduction vs. Star							2.1×	2.8×	3.3×

KV cache memory. Each token occupies 128 KB of KV cache. After Phase 1 KV discard, Star and Pulsar both retain only the local block, giving identical per-GPU KV cache:

$$M^{\text{KV}}_{\text{Star/Pulsar}} = \frac{L}{B} \times 128\,\text{KB}, \qquad M^{\text{KV}}_{\text{Dense}} = L \times 128\,\text{KB}. \tag{6}$$

At $L = 64$K: 2.15 GB per GPU (distributed) vs. 8.59 GB for Dense, a 4× per-GPU reduction exact and independent of s, k, or L.

Phase 1 FLOPs. FLOPs per layer for GQA attention are $\mathcal{F} = 2\,n^2\,(H_Q + H_{KV})\,d_h = 2\,n^2 \times 5{,}120$. Table 5 reports values at three context lengths.

Star Attention's constant 4× FLOPs reduction reflects its anchor always equalling one local block: $n^2_{\text{Star}}/n^2_{\text{Dense}} = 1/4$. Pulsar's growing advantage arises because the fixed prefix of 1,600 tokens represents a fraction of block size as L grows:

$$\frac{n^2_{\text{Star}}}{n^2_{\text{Pulsar}}} = \left(\frac{2L}{L + 6{,}400}\right)^2 \xrightarrow{L\to\infty} 4\times. \tag{7}$$

At $L = 64$K the ratio is already 3.3×.

Measured wall-clock latency and GPU memory. Table 6 reports end-to-end wall-clock speedup and Table 7 reports total GPU memory reduction over Dense, measured on Llama-3.1-8B-Instruct at 16K–128K.

Pulsar's speedup over Dense grows from 2× at 16K to 6.24× at 128K, widening with context, while Star grows more slowly (1.72× to 3.81×). This matches the analytical FLOPs estimates: Pulsar's fixed 1,600-token prefix is a diminishing fraction of block size as L grows (Table 5). Total fleet memory drops by 2.2× at 16K, rising monotonically to 5.1× at 128K; Star and Pulsar share identical measured peaks because both discard Phase 1 activations and retain only the local block KV cache. IDF table construction adds no measurable overhead, requiring only $O(L)$ integer operations on the CPU prior to any GPU computation.

Table 6: Wall-clock speedup over Dense on Llama-3.1-8B-Instruct ($B = 4$ for 16K–64K, $B = 8$ for 128K). Latency is the mean end-to-end time per sample averaged over all 13 RULER tasks. **Bold**: best per context length.

Method	**16K**	**32K**	**64K**	**128K**
Dense	—	—	—	—
Star	1.72×	2.24×	2.92×	3.81×
Pulsar	**2.00×**	**2.92×**	**4.27×**	**6.24×**

Table 7: Total GPU memory reduction over Dense on Llama-3.1-8B-Instruct ($B = 4$ for 16K–64K, $B = 8$ for 128K), summed across all GPUs (fleet cost). Star Attention is omitted because it retains identical measured memory to Pulsar after Phase 1 KV discard.

Method	**16K**	**32K**	**64K**	**128K**
Dense	—	—	—	—
Pulsar	2.2×	3.0×	4.0×	5.1×

6 Related Work

Distributed long-context inference. Sequence parallelism (Korthikanti et al., 2023) shards the sequence dimension with an all-reduce per layer. Ring Attention (Liu et al., 2023) removes this synchronization by overlapping communication with computation in a ring topology, but still requires cross-host transfers at every layer. Star Attention (Acharya et al., 2025) eliminates Phase 1 communication entirely by prepending a static first-block anchor to each host. Pulsar inherits Star's two-phase structure and communication-free Phase 1

but replaces the content-blind anchor with scored statistical summaries, recovering cross-block information without reintroducing communication.

Sparse and efficient attention. Longformer (Beltagy et al., 2020) combines sliding-window attention with global tokens at $O(n)$ cost but uses fixed structural patterns. MInference (Jiang et al., 2024a) dynamically selects sparse attention patterns per head during pre-filling, achieving strong results up to 64K but degrading at 128K in our experiments (Table 1). FlexPrefill (Lai et al., 2025) extends dynamic sparsity with query-dependent budget allocation; its gains are orthogonal to our distributed encoding phase. StreamingLLM (Xiao et al., 2023) preserves attention-sink tokens for streaming generation; we adopt its sink mechanism as one component of our prefix but pair it with content-aware summaries rather than a recency-based sliding window. FlashAttention (Dao et al., 2022) provides the tiled block computation underlying all our Phase 1 forward passes.

KV cache compression and context selection. H2O (Zhang et al., 2023), SnapKV (Li et al., 2024), and PyramidKV (Cai et al., 2024) evict low-importance KV entries during generation based on attention scores accumulated over prior tokens; these methods operate post-hoc on a fully computed KV cache and are orthogonal to Pulsar's Phase 1 selection. Because Pulsar's Phase 2 produces a standard distributed KV cache, these eviction strategies can be applied on top of Pulsar without modification, potentially compounding their savings with ours.

Prompt compression. LLMLingua (Jiang et al., 2023) and LongLLMLingua (Jiang et al., 2024b) use a small proxy LM to score and remove tokens from the prompt before the main forward pass, reducing input length at the cost of an additional neural inference step. RECOMP (Xu et al., 2023) similarly condenses retrieved passages via extractive or abstractive summarization. Pulsar's Max-IDF scoring is intentionally non-neural: it relies on corpus-level token statistics computable in a single CPU pass with no proxy model, adding negligible overhead while drawing on classical IDF-based retrieval (Robertson and Zaragoza, 2009). The key distinction is that Pulsar selects summaries *per block* for a distributed encoding phase, whereas prompt-compression methods target single-GPU prefill reduction.

7 Conclusion

We introduced Pulsar Attention, a drop-in replacement for Star Attention's static anchor block that uses two lightweight, content-aware components: a 64-token attention-sink prefix and Max-IDF block summaries at 12.5% of block size. On RULER with Llama-3.1-8B-Instruct, Pulsar outperforms both Dense and Star Attention at 32K–128K tokens, with gains of up to $+4.7\%$ over Dense and $+6.1\%$ over Star, while reducing Phase 1 per-GPU FLOPs by up to $3.3\times$ relative to Star. End-to-end wall-clock profiling and GPU memory measurements confirm that the analytical FLOPs advantage translates to practical speedups, with Pulsar achieving up to $6\times$ over Dense at 64K. Consistent gains on Qwen3-4B-Instruct confirm that the approach generalizes across model families. The retained KV cache is identical to Star Attention's, making Pulsar compatible with existing Phase 2 infrastructure and post-hoc KV compression methods. The main limitation is on tasks requiring retrieval of multiple values per key (NIAH MultiValue), where single-score chunk ranking can miss value-bearing chunks that lack globally rare tokens. Future work will address this through query-aware or multi-score selection strategies.

8 Limitations

While Pulsar Attention significantly reduces distributed encoding overhead, the MAX-IDF heuristic struggles to retrieve multiple recurring values associated with a single key (e.g., the NIAH MultiValue task). Because numerical values lack global token rarity, they receive low IDF scores and are often bypassed, necessitating future work on query-aware or multi-score selection strategies.

Second, our evaluation relies on synthetic benchmarks (RULER) and embedded reasoning suites (BABILONG). While rigorous, these do not fully capture the organic discourse structures and cross-document dependencies of real-world workloads, such as open-ended multi-document summarization or repository-level code reasoning.

Finally, the summary budget ($\sigma = 12.5\%$) and chunk size ($m = 32$) are static global hyperparameters. Because attention dynamics vary systematically by model depth, an adaptive, layer-wise budget allocation could achieve a more optimal trade-off between Phase 1 FLOPs reduction and exact global attention recovery.

References

Shantanu Acharya, Fei Jia, and Boris Ginsburg. 2025. Star attention: Efficient LLM inference over long sequences. In *Forty-second International Conference on Machine Learning (ICML)*.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and 1 others. 2024. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS 2022)*, volume 35, pages 16344–16359.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang He, Surin Garg, Qi Chen, Junkun Jiang, Yuqing Wei, Lili Chi, and 1 others. 2024a. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In *Advances in Neural Information Processing Systems (NeurIPS 2024)*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. Llmlingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pages 13358–13376.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024b. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1658–1677.

Vijay Anand Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. 2023. Reducing activation recomputation in large transformer models. *Proceedings of Machine Learning and Systems*, 5:341–353.

Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artem Sorokin, and Mikhail Burtsev. 2024. BABILong: Testing the limits of LLMs with long context reasoning-in-a-haystack. *arXiv preprint arXiv:2406.10149*.

Xunhao Lai, Jianqiao Lu, Yao Luo, Yiyuan Ma, and Xun Zhou. 2025. Flexprefill: A context-aware sparse attention mechanism for efficient long-sequence inference. *arXiv preprint arXiv:2502.20766*.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970.

Hao Liu, Matei Zaharia, and Pieter Abbeel. 2023. Ring attention with blockwise transformers for near-infinite context. *arXiv preprint arXiv:2310.01889*.

Maxim Milakov and Natalia Gimelshein. 2018. Online normalizer calculation for softmax. *arXiv preprint arXiv:1805.02867*.

Qwen Team. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 2383–2392.

Stephen Robertson and Hugo Zaragoza. 2009. *The probabilistic relevance framework: BM25 and beyond*, volume 4. Now Publishers Inc.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2023. Recomp: Improving retrieval-augmented lms with compression and selective augmentation. *arXiv preprint arXiv:2310.04408*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

A Dataset Details

A.1 BABILong

BABILong (Kuratov et al., 2024) extends the bAbI reasoning suite by embedding short synthetic reasoning chains inside long distractor text drawn from PG-19 books. The reasoning chains are generated from symbolic rules; the model must locate the relevant sentences among thousands of distractor tokens and apply multi-step logic. We evaluate on two tasks; Table 8 summarises their requirements.

Context splits cover $L \in \{16\text{K}, 32\text{K}, 64\text{K}, 128\text{K}\}$ tokens; each split contains 1,000 samples. We report results for `qa1`, `qa3`, and `qa5` as representative of increasing reasoning depth.

A.2 RULER

RULER (Hsieh et al., 2024) is a synthetic benchmark designed to test long-context recall under controlled conditions. We use all 13 tasks across four context lengths ($L \in \{16\text{K}, 32\text{K}, 64\text{K}, 128\text{K}\}$), 500 samples per task per length. Table 9 describes each task.

B Chat Template and Prompt Format

B.1 Llama 3 Template

Llama experiments use Meta-Llama-3.1-8B-Instruct with the standard Llama 3 instruct chat template. The full template skeleton is:

```
<|begin_of_text|>
<|start_header_id|>system<|end_header_id|>

You are a helpful assistant.<|eot_id|>
<|start_header_id|>user<|end_header_id|>

{task_instruction}

{in_context_examples}

{post_prompt}

<context>
{long_context}
</context>

Question: {query}<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
```

Stop words (Llama 3). Generation halts at any of: `<|end_of_text|>`, `<|eom_id|>`, `<|eot_id|>`, or after `max_new_tokens` (128 for BABILong; task-dependent for RULER, see Table 11).

Tokenisation. Tokens are produced with `add_special_tokens=False`; the `<|begin_of_text|>` BOS token is included as part of the template string itself and must not be added a second time.

B.2 Qwen3 Template

Qwen3 experiments use Qwen3-4B-Instruct (Qwen Team, 2025) with the ChatML instruct template. Thinking mode is disabled throughout all experiments. The full template skeleton is:

```
<|im_start|>system
You are a helpful assistant.<|im_end|>
<|im_start|>user

{task_instruction}

{in_context_examples}

{post_prompt}

<context>
{long_context}
</context>

Question: {query}<|im_end|>
<|im_start|>assistant
```

Stop words (Qwen3). Generation halts at: `<|im_end|>` or `<|endoftext|>`, or after `max_new_tokens` (same per-task values as Table 11).

Tokenisation. As with Llama 3, tokens are produced with `add_special_tokens=False`; the system header is embedded directly in the template string.

B.3 Context/Query Split

Both models share the same two-string decomposition before tokenisation:

- **`prompt_context`**: everything from the opening header token through the closing `</context>` tag, inclusive. This string is tokenised, split into blocks of size B, and processed by Phase 1.
- **`prompt_query`**: the `Question: {query}` fragment followed by the assistant header tokens. This string is tokenised separately and processed by Phase 2.

C Scoring Heuristic Details

All heuristics operate on non-overlapping contiguous chunks of `chunk_size` tokens carved from each block. Chunks preserve local word order; the top-`num_chunks` chunks by score are selected and returned *in positional order* to maintain natural reading order for the Transformer.

Table 8: BABILong task descriptions.

Task	Description
qa1	**Single Supporting Fact.** One sentence states where a person currently is. The model must track the most recent location across multiple movement facts and answer "Where is X?"
qa3	**Three Supporting Facts / Object Path.** Three facts establish an object's path across locations. The model must answer "Where was the object *before* location Y?", requiring three-hop reasoning over the chain.
qa5	**Three-Argument Relations.** Facts involve object-transfer events (picked-up, gave, passed). The model answers queries such as "Who gave the apple?" or "What did X give to Y?", requiring entity-argument co-reference across three-place predicates.

Table 9: RULER task descriptions. Abbreviations: k = number of unique needle keys; v = values per key; r = number of queries.

Category	Task	Description
NIAH	Single-1	Single word key → single number value hidden in synthetic noise text.
	Single-2	Single word key → single number value hidden in Paul Graham essay excerpts.
	Single-3	Single word key → UUID value hidden in book text. UUID tokens are maximally rare (high IDF).
	MultiKey-1	4 distinct word keys all map to the same number; the model must retrieve the value from any key.
	MultiKey-2	An entire line of word keys maps to one value; model must match the full key phrase.
	MultiKey-3	UUID keys → UUID values; both key and value are rare tokens.
	MultiValue	1 key maps to 4 different numbers; the model must recall all of them.
	MultiQuery	4 queries each asking for the value of a different key from the same set of needles.
Tracking	Variable	A chain of variable assignments is hidden in noise (e.g., `x=3; y=x; z=y`). The model finds all variables assigned a target value.
Words	CWE	A numbered list of words is presented; some appear more often. The model identifies the 10 most common words.
	FWE	A "coded" text with high-frequency synthetic words. The model identifies the 3 most frequent codes.
QA	QA1	Reading comprehension over SQuAD (Rajpurkar et al., 2016) passages concatenated into a long context.
	QA2	Multi-hop reading comprehension over HotpotQA (Yang et al., 2018) documents; requires reasoning across two documents.

C.1 Corpus-Level IDF Table

Before any scoring, a single pass over all blocks constructs a document-frequency table. Let N be the total number of blocks and $\mathrm{df}(t)$ the number of blocks containing token t:

$$\mathrm{IDF}(t) = \log\left(\frac{N}{\max(\mathrm{df}(t),\, 1)}\right). \tag{8}$$

This is stored as a dense integer-indexed tensor `idf_table` of shape `(vocab_size,)`. All four IDF-based heuristics perform a single gather from this table; no Python loops over the vocabulary are required.

C.2 TF-IDF

$$\mathrm{score}_{\text{TF-IDF}}(C) = \frac{1}{|C|}\sum_{p\in C}\mathrm{IDF}(t_p). \tag{9}$$

This equals the classical mean TF-IDF over types, and permits a single vectorized lookup: `idf_table[chunk_tokens].sum() / len(chunk)`.

Strengths. Balanced, general-purpose; promotes chunks with rare but recurrent vocabulary. **Weakness.** A single rare entity among many common tokens is diluted by the average.

C.3 BM25

$$\mathrm{score}_{\text{BM25}}(C) = \sum_{t\in\text{types}(C)} \mathrm{IDF}_{\text{BM25}}(t) \times \frac{\mathrm{tf}(t)\,(k_1+1)}{\mathrm{tf}(t)+k_1 D(C)}, \tag{10}$$

where $D(C) = 1 - b + b\,\frac{|C|}{\bar{l}}$, $k_1 = 1.2$, $b = 0.75$, $\bar{l}$ is the mean chunk length across the corpus, and

$$\mathrm{IDF}_{\text{BM25}}(t) = \log\left(\frac{N-\mathrm{df}(t)+0.5}{\mathrm{df}(t)+0.5}+1\right).$$

Strengths. More robust than TF-IDF when chunk lengths vary. **Weakness.** Slightly higher compute; benefits diminish when all chunks are the same length.

C.4 Entropy (Type-Token Ratio)

$$\text{score}_{\text{Entropy}}(C) = \frac{\left|\{t : t \in C\}\right|}{|C|}. \quad (11)$$

Implemented as `(bincount(C) > 0).sum() / len(C)`.

Strengths. Corpus-independent; no IDF table required. A chunk of padding tokens or repeated stop-words scores near 0; an entity-rich chunk scores near 1. **Weakness.** Ignores global rarity; a chunk with 32 distinct common stop-words scores identically to one with 32 rare entities.

C.5 Max-IDF

$$\text{score}_{\text{Max-IDF}}(C) = \max_{p \in C} \text{IDF}(t_p). \quad (12)$$

Implemented as `idf_table[chunk_tokens].max()`.

Strengths. Acts as a pure needle detector: if any token in the chunk appears in only one block, the chunk is guaranteed to be selected. This aligns with what multi-hop tasks and NIAH require. **Weakness.** Ignores overall chunk quality; one rare token among 31 uninformative tokens still wins.

C.6 Even-Spaced (Baseline)

Selects exactly `num_chunks` $\times$ `chunk_size` tokens at uniform intervals across the block using `torch.linspace`:

$$\begin{aligned} \text{indices} &= \lfloor \texttt{linspace}(0,\ |B| - 1,\ s_{\text{sink}}) \rfloor, \\ \sigma &= \texttt{num_chunks} \times \texttt{chunk_size}. \end{aligned} \quad (13)$$

Original (sparse) position IDs are preserved.

Strengths. No corpus statistics required; deterministic. **Weakness.** Wastes budget on low-information spans as budget grows (confirmed in ablations: `qa3` degrades from 36 to 25 as σ increases from 128 to 1024).

C.7 Heuristic Summary

Table 10 consolidates the five heuristics described above, listing the signal each one uses, whether it requires corpus-level statistics, the task profile it suits best, and its main weakness.

D Hyperparameter Configuration

Table 11 lists all hyperparameters used in our main experiments. All values are fixed across RULER and BABILong unless noted.

E RoPE Position ID Assignment

When a block's Phase 1 input is assembled as $[\text{SINK} \mid S_1, \ldots, S_{i-1} \mid B_i]$, the position IDs of the inserted sink and summary tokens can be assigned in two ways.

Sparse (default). Every token retains its *original global position* in the full context. Summary tokens from block j keep the position IDs of their source locations. The assembled sequence therefore contains *gaps* where non-selected tokens were dropped. This preserves the true inter-token distances that RoPE was trained to encode: a summary token from block 0 is still seen as far from a token in block 3, matching the model's pre-training geometry.

Contiguous. The assembled sequence is renumbered $0, 1, \ldots, L_i - 1$ for each block i's Phase 1 forward pass (original Star Attention style). In this mode, Phase 2 query position IDs are offset to start just past the longest assembled Phase 1 length, ensuring RoPE deltas from the query to every block's KV remain small and in-distribution.

Contiguous mode can occasionally help by ensuring that summary tokens and local block tokens are seen at short relative distances, but it distorts the global geometry. All experiments reported in this paper use **sparse** mode, which aligns with the model's pre-training distribution.

F KV Cache Discard

After each Phase 1 forward pass, the model's KV cache contains entries for *all* tokens in the assembled input $[\text{SINK} \mid \Sigma_0, \ldots, \Sigma_{i-1} \mid B_i]$. The sink and summary KV states are discarded immediately:

$$\begin{aligned} K_i^{(\text{layer})} &\leftarrow K_i^{(\text{layer})}[\,:, :, \texttt{is_bi}\,], \\ V_i^{(\text{layer})} &\leftarrow V_i^{(\text{layer})}[\,:, :, \texttt{is_bi}\,], \end{aligned} \quad (14)$$

where $\texttt{is_bi}[p] = \text{True}$ iff $p \geq \texttt{summary_len}$. Implemented via a boolean mask:

```
is_bi = torch.zeros(total_len, dtype=torch.bool)
is_bi[-block_size:] = True
kv = [[x[0][:,:,is_bi], x[1][:,:,is_bi]] for x in kv]
```

Discarding serves two purposes: **(1)** *No duplicate tokens in Phase 2*: summary token S_j was prepended to blocks $j+1, \ldots, B-1$; if retained, each KV cache would contain a copy of σ_j's key/value vectors, corrupting the softmax normalisation during Phase 2's `all_gather`. **(2)** *Exact final cache size*: after discarding, each GPU holds

Table 10: Summary of all scoring heuristics.

Heuristic	Signal	Corpus-dep.	Best for	Weakness
TF-IDF	Mean IDF	Yes	General-purpose	Single rare token diluted
BM25	Len-norm. IDF	Yes	Variable chunk sizes	Extra `bincount` cost
Entropy	Type-token ratio	No	Small corpus / fast	Blind to global rarity
Max-IDF	Peak IDF	Yes	Rare entity / NIAH	Ignores chunk quality
Even-Sp.	Uniform coverage	No	Coverage guarantee	Wastes budget on filler

Table 11: Full hyperparameter configuration.

Parameter	Value	Notes
Model — Llama 3		
Model	Meta-Llama-3.1-8B-Instruct	
Precision	bfloat16	
Attention impl.	FlashAttention-2	`attn_impl=flash_attention_2`
Decoding	Greedy (argmax)	Temperature = 0
Model — Qwen3		
Model	Qwen3-4B-Instruct (Qwen Team, 2025)	
Precision	bfloat16	
Attention impl.	FlashAttention-2	GQA: 32 Q-heads / 8 KV-heads
Thinking mode	Disabled	
Decoding	Greedy (argmax)	Temperature = 0
Distributed setup		
Number of GPUs (H)	4	B blocks split evenly between hosts
Block size	25% of context length	12.5% of length at 128K for memory efficiency
Pulsar Attention: summary		
Summary method	Max-IDF	Eq. (12)
Summary budget (σ)	512 tokens / block (at 16K)	12.5% of block size
Chunk size	32 tokens	16 chunks selected per block
Sink size (s_{sink})	64 tokens	First 64 tokens of block 0
KV discard	Enabled	Non-B_i KV sliced after Phase 1
Position mode	Sparse	Original global position IDs preserved
Generation		
Max new tokens (BABILong)	128	
Max new tokens (RULER NIAH)	128	
Max new tokens (RULER VT)	30	
Max new tokens (RULER CWE)	120	
Max new tokens (RULER FWE)	50	
Max new tokens (RULER QA)	32	

exactly `block_size` KV entries, so the total KV cache across B GPUs equals the original context length L.

G Extended BABILong Heuristic Results

Tables 12 and 13 extend the 16K heuristic comparison (Table 4 in the main paper) to 32K and 64K context lengths, at both the 3.125% ($\sigma = 128$) and 12.5% ($\sigma = 512$) summary budgets ($B = 4$, block sizes 8K and 16K respectively). At the 12.5% budget ($s = 512$), Max-IDF is best on multi-hop tasks (`qa3`, `qa5`) across both 32K and 64K. At the smaller 3.125% budget ($\sigma = 128$), averaging heuristics (TF-IDF, BM25) lead on `qa3`: at this budget the fixed chunk count is small enough that selecting diverse representative chunks outweighs pure needle-detection. Max-IDF dominates `qa5` at 32K–3.125% but not at 64K–3.125%, where `qa5` scores are uniformly lower and BM25/Entropy tie for best. Star Attention was not evaluated at 64K and its cells are left blank.

H Phase 2 Distributed Softmax Merge

Phase 2 runs an identical forward pass on all B GPUs simultaneously. Each GPU computes a local attention output $\mathbf{o}_h$ and log-sum-exp ℓ_h over its own KV block. After `dist.all_gather`, the outputs are merged with the numerically stable online-softmax formula (Milakov and Gimelshein, 2018):

$$\ell' = \ell + \log\left(1 + e^{\ell_h - \ell}\right), \tag{15}$$

$$\mathbf{o}' = e^{\ell - \ell'}\,\mathbf{o} + e^{\ell_h - \ell'}\,\mathbf{o}_h, \tag{16}$$

Table 12: BABILONG-32K accuracy (%) across two summary budgets ($B = 4$, block size 8 K). **Bold**: best among the four statistical heuristics per group. Star Attention uses a fixed anchor block independent of summary budget.

	$\sigma = 128$ (3.125%)			$\sigma = 512$ (12.5%)		
Heuristic	qa1	qa3	qa5	qa1	qa3	qa5
TF-IDF	**34**	**17**	**48**	**32**	15	48
BM25	28	**17**	45	29	17	46
Entropy	21	15	42	30	15	45
Max-IDF *(ours)*	28	10	43	31	**19**	**49**
Even-Spaced	33	17	55	30	16	48
Star Attn	28	13	41	28	13	41

Table 13: BABILONG-64K accuracy (%) across two summary budgets ($B = 4$, block size 16 K). **Bold**: best among the four statistical heuristics per group.

	$\sigma = 128$ (3.125%)			$\sigma = 512$ (12.5%)		
Heuristic	qa1	qa3	qa5	qa1	qa3	qa5
TF-IDF	**28**	9	35	17	5	31
BM25	26	**10**	**38**	15	4	30
Entropy	22	8	**38**	20	6	36
Max-IDF *(ours)*	22	7	33	**28**	**11**	**40**
Even-Spaced	28	2	56	24	5	34

applied iteratively over all gathered outputs. Equations (15)–(16) are implemented as a `torch.jit.script` kernel for efficiency. The result is mathematically equivalent to running full attention over the concatenated KV cache of all B GPUs, at the cost of one `all_gather` communication round per generation step.

I Efficiency Measurement

All timing and peak-memory experiments are conducted on a single node equipped with 4 × A100-80 GB GPUs interconnected via NVLink, using bfloat16 precision and FlashAttention-2.

Wall-clock speedup. For each configuration (dense full attention, Star Attention, Pulsar Attention), N generation samples are processed sequentially. Per-sample time is measured with a high-resolution monotonic clock started immediately before the forward pass and stopped after a cross-rank synchronisation barrier, ensuring that the recorded time reflects the slowest GPU (i.e. the true end-to-end latency). Speedup is reported as the ratio of the mean per-sample wall-clock time under dense attention to the mean per-sample time under the respective method. Dataset loading, tokenisation, and IDF-table construction are excluded from all timings.

Peak VRAM. Per-device peak memory statistics are reset before each sample. After generation completes, peak GPU memory is read from the device allocator, converted to gigabytes, and recorded per rank. We report two aggregates:

- **Per-GPU peak**: maximum recorded peak across all B ranks, reflecting the most memory-pressured device.
- **Total peak**: sum of per-rank peaks across all B GPUs, reflecting the aggregate KV-cache footprint of the full context.

For dense attention (single-process, multi-GPU via `device_map=auto`), the total peak is computed as the sum over all visible CUDA devices on the node, ensuring comparability with the distributed total for the other methods.