Proceedings of CONF-SEML 2026 Symposium: Computational Analysis and Modeling in Complex Intelligent Systems
DOI: 10.54254/2755-2721/2026.GU33803

Architectural Constraints in Long-Sequence LLM Training: An Analytical Perspective on Hardware–Software Co-Design

Zhenhuan Shao

Glasgow College, University of Electronic Science and Technology of China, Chengdu, China
2960315@student.gla.ac.uk

Abstract. With Large Language Models (LLMs) moving toward ultra-long contexts, limitations in memory capacity, bandwidth, and inter-device communication increasingly restrict training efficiency. Long-context modeling exacerbates the already high resource demands of LLMs, as both the attention mechanism and the surrounding feed-forward and normalization layers scale with sequence length. This paper analyzes long-sequence LLM training from operator-level, system-level, and hardware-level perspectives. Specifically, it highlights that IO-aware exact-attention kernels can significantly reduce HBM traffic for attention computation, but they do not mitigate the activation-memory growth in non-attention modules. When sequence length grows, the main bottlenecks become communication, synchronization, and workload imbalance in distributed sequence/context parallelism. Importantly, the exact crossover point is model-, hardware-, and implementation-dependent. Future scaling will likely demand a combination of software optimization and co-design of memory systems, packaging, and interconnects.

Keywords: Long-Sequence Training, Memory Wall, Hardware-Software Co-design, Sequence Chunking

1. Introduction

In long-sequence Large Language Models (LLMs) training, the dominant bottleneck does not remain fixed. Specifically, IO-aware exact-attention kernels substantially reduce HBM traffic, whereas model-wide memory control methods such as MST and BPT shift the constraint toward saved activations and temporary states [1-5]. In this context, distributed sequence and context parallel systems such as Ring Attention, DeepSpeed-Ulysses, USP, Striped Attention, WLB-LLM, DCP, and FPDT further increase the feasible context length, while revealing communication, synchronization, and workload-balance limitations [6-12]. Notably, exact attention refers to mathematically precise self-attention rather than sparse or low-rank approximations, and additionally, sequence/context parallelism denotes distributing sequence states across devices to reduce per-device memory usage. By reviewing the existing literature, this paper treats long-context training as a bottleneck-migration phenomenon across operator, system, and hardware levels, rather than merely as an attention-kernel problem or a universal crossover rule. Consequently, this work helps to systematically understand the multi-level challenges of long-context training and provides guidance for designing more efficient model-hardware co-optimization strategies.

© 2026 The Authors. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

DOI: 10.54254/2755-2721/2026.GU33803

2. Fundamental physical constraints in long-sequence training

2.1. The physical compute-bandwidth mismatch

The compute-bandwidth relationship can be characterized by two key quantities, the machine balance, $\beta_{mach}=\frac{\Pi_{peak}}{B_{HBM,peak}}$, and the workload operational intensity, $I_{workload}=\frac{FLOPs}{Bytes\ moved}$. As a result, kernels with operational intensity below the machine balance are likely to become memory-sensitive, as $I_{workload}$ is lower than β_{mach}. Therefore, peak hardware specifications alone do not prove that training is memory-bound; they only indicate that future kernels must sustain increasingly high arithmetic intensity to fully utilize compute resources [13, 14]. In practice, long-sequence training involves both local HBM traffic and global inter-device communication, so the actual bottleneck depends on both operator efficiency and distributed communication efficiency [7, 8, 10, 11].

To clarify these quantities, Π_{peak} is peak compute throughput, $B_{HBM,peak}$ represents peak HBM bandwidth, and Bytes moved is total memory traffic. As shown in Table 1, the A100 and H100 GPUs provide around 153 and 295 Ops/Byte, respectively [13, 14]. This comparison shows that kernels whose operational intensity does not scale accordingly become more memory-sensitive, motivating I/O-aware exact attention [1-3].

Table 1. Representative machine characteristics relevant to long-context training

Tensor GPU Architecture	Peak Dense FP16, (TFLOPS)	HBM Bandwidth (GB/s)	Machine Balance Ops:Byte Ratio	Intra-node Interconnect Bandwidth (NVLink, bidirectional GB/s)	Inter-node Network Bandwidth (InfiniBand, Unidirectional GB/s)
A100 SXM 80GB [13]	312	2039	≈153	600	25
H100 SXM [14]	989	3352	≈295	900	50

Note: Machine balance is a hardware characteristic derived from peak compute throughput and HBM bandwidth. It is not, by itself, proof that a real workload is memory-bound. For H100, the value 989 TFLOPS is used as a dense/no-sparsity estimate for consistency with the machine-balance calculation; vendor pages may also report higher sparsity-enabled FP16/BF16 Tensor Core throughput.

2.2. Activation-induced memory bloat

Beyond attention, long-sequence training is constrained by peak memory consumption. In particular, the MLP intermediate width is given by:

$$d_{ff} = \alpha \bullet H \tag{1}$$

where d_{ff} is the feed-forward intermediate width, H is the hidden dimension, and α is the expansion ratio. Under 16-bit precision, a simplified SwiGLU sub-model has $M_{MLP_Act}=2\times(3\times B\times S\times d_{ff}\times L)$ Bytes, where B is micro-batch size, S is sequence length, and L is layer count. As such, MST reports 12-24× longer trainable context without throughput or convergence degradation [4].

In LLaMA-style models, the MLP activation term grows linearly with S, L, and B. Once attention memory is compressed, this term becomes increasingly important [4]. In order to capture this effect, peak memory can be decomposed as:

DOI: 10.54254/2755-2721/2026.GU33803

$$M_{peak} = Mparam + Mopt + Mgrad + Msaved_act(S) + Mattn_buf(S,P,C) + Mcomm_buf(P) + Mmisc \quad (2)$$

where Mparam , Mopt , and Mgrad denote parameter, optimizer-state, and gradient memory; Msaved_act(S) represents saved activations; Mattn_buf (S,P,C) refers to attention-side temporary buffers that depend on sequence length S, parallel degree P, and chunk size C; Mcomm_buf (P) is communication buffers; and Mmisc denotes other workspace overhead. Unless otherwise stated, this equation describes peak per-device memory under distributed execution [4, 12].

In particular, this decomposition separates parameter, optimizer, gradient, saved activation, attention, and communication memory, giving a clearer view than a single activation alone. MST shows that after attention memory is reduced, MLP and LM-head intermediates become a major limiter of maximum trainable sequence length [4]. Thus, operator-level attention optimization is necessary but not sufficient, as long-sequence training is ultimately constrained by the combined growth of saved activations, temporary states, and distributed communication buffers [4, 12].

3. Hardware-aware optimization and global memory control for exact attention

3.1. I/O-awareness and tiling mechanisms

The efficiency of exact attention is primarily limited by memory bandwidth, making I/O-aware tiling mechanisms crucial for long-sequence training. Specifically, standard self-attention materializes large intermediate tensors, particularly the score matrix and its downstream softmax-related states, which heavily stress HBM. To address this bottleneck, FlashAttention tiles the attention computation and keeps blocks in SRAM, avoiding the need to write back the full attention matrix to HBM [1]. Building on this approach, FlashAttention-2 and FlashAttention-3 further improve practical performance through more effective work partitioning, increased occupancy, and hardware-specific scheduling on modern GPU architectures [2, 3].

3.2. Attention I/O complexity and asymptotic optimization

The computational cost of exact self-attention stays quadratic in sequence length, but the dominant HBM I/O pattern changes once tiling and recomputation are applied. FlashAttention drives the main shift in I/O, while FlashAttention-2 and FlashAttention-3 mainly boost constant factors and hardware use rather than altering the asymptotic complexity [1-3, 15, 16]. In detail, standard attention materializes large intermediate tensors, causing heavy HBM traffic, which FlashAttention-1 mitigates through I/O-aware tiling and recomputation. FlashAttention-2 preserves the same asymptotic complexity while improving work partitioning and occupancy, and FlashAttention-3 further boosts hardware utilization with Hopper-oriented asynchrony and low-precision support [1-3, 16]. Table 2 shows these operator-level I/O evolutions, distinguishing asymptotic I/O reduction from practical hardware improvements.

DOI: 10.54254/2755-2721/2026.GU33803

Table 2. Operator-level I/O evolution of exact self-attention

Attention Mechanism	FLOPs Complexity	Dominant HBM I/O Pattern	Main Practical Contribution
Standard Attention [16]	$O(S^2d)$	Large intermediate tensors	Simple but heavy HBM traffic
FlashAttention-1 [1]	$O(S^2d)$	Tiled attention; reduced HBM traffic	IO-aware tiling and recomputation
FlashAttention-2 [2]	$O(S^2d)$	Same asymptotic class as FA-1	Better work partitioning and occupancy
FlashAttention-3 [3]	$O(S^2d)$	Same asymptotic class as FA-1	Hopper-oriented asynchrony and low-precision support

Note: S = sequence length; d = dimension of the attention head. The table distinguishes asymptotic I/O reduction from practical hardware-utilization improvements.

3.3. System-level memory limitations of operator optimizations

Although FlashAttention and its variants substantially reduce attention-side memory traffic, they do not alleviate the memory footprint of saved activations in non-attention modules. Thus, as sequence length approaches 100K tokens and beyond, the relative impact of locally optimized attention can diminish in configurations where attention-side temporary memory has already been reduced, while MLP/FFN activations and distributed communication buffers may become increasingly significant, while simultaneously the contributions of MLP/FFN activations and distributed communication buffers grow disproportionately [4, 5, 12]. This shift implies that memory pressure in long-sequence training is no longer dominated by attention kernels alone but arises from the combined scaling of intermediate activations and inter-device buffers. Therefore, addressing long-sequence training effectively requires a holistic memory-management strategy that considers both operator-level optimizations and system-level memory interactions, rather than treating attention optimization in isolation

4. System-level memory and communication constraints in global sequence chunking

4.1. Sequence chunking and physical topology evolution

Long-context training systems differ mainly in how they partition the sequence dimension and organize inter-device communication. For example, MST uses single-GPU temporal chunking, BPT and Ring Attention use blockwise or ring-based partitioning, DeepSpeed-Ulysses redistributes Q, K, and V via all-to-all communication, and USP merges Ring- and Ulysses-style strategies [4-8]. Although these methods differ in whether they operate on a single GPU or across multiple GPUs, they share a common objective: reducing the amount of sequence-dependent intermediate state that must be materialized or retained at once.

At a finer level, Ring-style parallelism employs overlapping point-to-point exchanges of sequence blocks, while Ulysses uses all-to-all redistribution around attention. Accordingly, Ulysses

DOI: 10.54254/2755-2721/2026.GU33803

benefits most from high collective bandwidth, while Ring-style methods are more sensitive to overlap and load-balance efficiency [6-8]. Thus, the per-device distributed-memory footprint can be expressed as:

$$Mpeak(dist) = Mparam(local) + Mopt(local) + Mgrad(local) + \frac{Msaved_act(S)}{P} + Mrepartition + Mcomm_buf \quad (3)$$

where $\mathrm{Mparam(local)}$, $\mathrm{Mopt(local)}$, and $\mathrm{Mgrad(local)}$ are per-device parameter, optimizer, and gradient footprints; $\frac{\mathrm{Msaved_act(S)}}{\mathrm{P}}$ is the reduced sequence-state memory under P-way partitioning; $\mathrm{Mrepartition}$ denotes temporary states created by sequence/head redistribution; and $\mathrm{Mcomm_buf}$ the extra communication buffer [6-8, 10-12]. This serves as a comparative trend, not a strict upper bound. It highlights the key trade-off: local sequence-state memory drops with partitioning, while repartition and communication costs rise.

4.2. Peak memory and I/O complexity analysis

Under sequence partitioning, per-device peak memory is lowered because each device stores only part of the sequence state. However, the net gain depends on both local-memory relief and communication efficiency. DeepSpeed-Ulysses reduces per-device sequence memory through all-to-all redistribution before and after attention, while Ring Attention relies on overlapped point-to-point communication [6, 7]. Under grouped-query or multi-query attention, practical flexibility is further constrained by the head-related structure of the model, which is why USP treats Ring-style and Ulysses-style sequence parallelism as complementary rather than mutually exclusive [8]. Table 3 offers a comparative view of systems, not a normalized benchmark. The last row groups several recent frameworks with different mechanisms, without implying they are interchangeable.

Table 3. Classification of global sequence partitioning and GPU memory control architectures

Architecture	Main strategy	Per-device memory trend	Communication pattern	Load-balance property
MST [4]	Temporal chunking on a single GPU	Bounded by chunk size C	No inter-GPU communication	N/A; no inter-GPU load imbalance
Ring Attention [6]	Spatial partitioning with ring exchange	Approx. O(S/P) + buffers	P2P ring communication	Imbalanced under causal masking
DeepSpeed-Ulysses [7]	Sequence partition + all-to-all head redistribution	Approx. O(S/P) + all-to-all buffers	All-to-all collectives	Good when bandwidth is high
USP [8]	Hybrid Ring + Ulysses sequence parallelism	Topology-adaptive reduction	Hybrid collective + P2P	Improved flexibility
Striped Attention [9]	Redistributed token placement for Ring-style execution	Same order as Ring	P2P ring communication	Improved under causal masking
Representative system-level methods [10-12]	Dynamic balancing, context parallelism, and pipelined distributed execution	System-dependent but reduced by partitioning	Mixed collectives / pipeline communication	Improved under irregular workloads

DOI: 10.54254/2755-2721/2026.GU33803

Note: P = number of parallel devices; C = chunk size. The memory expressions are simplified comparative forms rather than universal upper bounds.

4.3. Physical limits of communication-computation overlap

In long-sequence training, communication-computation overlap is constrained by system topology and workload. Although chunking of sequences reduces memory use, it does not eliminate communication costs [6-12]. Moreover, step-time modeling shows that total step time depends on the larger of compute or communication time, in addition to launch and synchronization overhead. In particular, all-to-all redistribution schemes, like Ulysses, expose collective communication overhead, whereas Ring-style exchanges may suffer from load imbalance under causal masking [6, 7]. To address this, redistribution of sequence blocks in Striped Attention mitigates these issues and achieves more uniform workload distribution [9]. Moreover, empirical measurements demonstrate that WLB-LLM reaches up to 1.44× compute-latency skew across GPUs and DCP achieves 1.19-2.45× attention acceleration under causal masks [10, 11]. Consequently, communication-computation overlap should be treated as a workload- and topology-dependent property rather than as a threshold determined solely by bandwidth [7, 9-12].

4.4. Micro-architectural trade-offs and throughput degradation

Micro-architectural trade-offs define the limits of memory reduction via sequence chunking. Although chunking can decrease local memory, its benefit diminishes once implementation overhead becomes significant. Moreover, extremely small chunks reduce effective GPU utilization because unfavorable matrix shapes, lower occupancy, and poorly amortized launch or synchronization overhead collectively degrade performance [2, 5, 15]. Consequently, chunk size emerges as a performance-sensitive parameter rather than a free variable. Further memory reduction through finer chunking or more aggressive repartitioning incurs trade-offs, including lower kernel efficiency, increased synchronization overhead, and a worsened communication-to-compute balance [5, 10, 11]. These interactions collectively establish a practical trade-off frontier under current GPU architectures.

5. Global optimization and hardware constraints

5.1. Limits of space-time trade-offs in global optimization

While global optimization reduces local bottlenecks, it usually leads to new spatio-temporal trade-offs. For example, FlashAttention alleviates local memory pressure by reducing the HBM access traffic of the attention module, but this optimization depends on fast high-bandwidth memory access, and its time efficiency remains constrained by bus bandwidth and computation scheduling [1]. Similarly, MST and BPT reduce memory usage by saving activations or using block-level execution, but they introduce extra control logic and memory access sequences, increasing latency and synchronization [4, 5]. Other methods, like Ring, Ulysses, USP, Striped Attention, WLB-LLM, DCP, and FPDT, decrease per-device memory usage through sequence or context partitioning. While this strategy improves spatial efficiency, it also raises cross-device communication costs and load imbalance, posing additional challenges for global scheduling and bandwidth management [6-12]. These methods show typical spatio-temporal trade-offs: saving memory often raises time costs, while speeding up computation can increase memory pressure, limiting software-level gains.

DOI: 10.54254/2755-2721/2026.GU33803

5.2. Hardware co-design and memory wall breakthroughs

As software-level optimization yields diminishing returns, hardware parameters grow more important. In particular, advanced 3D integration and hybrid bonding can enhance fast-memory capacity and local bandwidth, which improves memory-system conditions for sequence chunking and operator tiling [17]. For example, memory-centric architectures like PIM or NMC can reduce data movement for specific Transformer operations. TransPIM demonstrates that collaborative software-hardware design based on dataflow and memory-side processing can accelerate Transformers, though reduction and Softmax-like operations remain challenging [19]. At the cluster level, SiP-ML suggests that training efficiency may be limited by network topology and effective bandwidth [20]. In contrast, inference-focused research such as H2-LLM serves only as an architectural analogy, since training also involves backpropagation, optimizer states, gradient flow, and distributed synchronization [18]. At a more detailed level, advanced packaging technologies can increase fast-memory capacity and local bandwidth, while interconnect co-design affects the effective communication bandwidth β_{eff} and startup overhead. In addition, memory-centric computing also reduces data movement for specific operations. For example, experiments with TransPIM speeds up memory-accelerated operations by 3.7–9.1×, and SiP-ML improves training time by 1.3-9.1× under optical interconnect co-design [19, 20]. These hardware enhancements provide the system with greater memory and bandwidth support, but they do not directly address the challenges of long-context training. Meanwhile, H2-LLM is primarily used as an architectural analogy, since actual training also involves backward propagation, optimizer states, and distributed gradient flow [17, 18].

6. Conclusion

This study examines the architectural constraints of long-sequence LLM training from operator, system, and hardware perspectives. Long-context scaling is driven by bottleneck migration rather than a single fixed limit. While IO-aware exact-attention kernels can reduce local HBM traffic, the memory footprint of activations from non-attention modules are still large. Similarly, sequence and context parallelism can extend feasible context lengths by lowering per-device memory requirements, but this comes at the cost of communication overhead, synchronization, and workload imbalance. In addition, hardware factors like fast-memory capacity, effective bandwidth, packaging density, and interconnect efficiency become increasingly critical. Consequently, longer-context training requires a layered co-design across kernels, distributed systems, memory hierarchies, and hardware, with exact crossover points depending on the model, hardware, and implementation.

References

[1] Dao, T., Fu, D. Y., Ermon, S., Rudra, A., & Ré, C. (2022). FlashAttention: Fast and memory-efficient exact attention with IO-awareness. Advances in Neural Information Processing Systems (NeurIPS), 35, 16344-16359.
[2] Dao, T. (2024). FlashAttention-2: Faster attention with better parallelism and work partitioning. In Proc. 12th International Conference on Learning Representations (ICLR).
[3] Shah, J., et al. (2024). FlashAttention-3: Fast and accurate attention with asynchrony and low-precision. arXiv preprint arXiv: 2407.08608.
[4] Luo, C., et al. (2024). Mini-sequence transformer: Optimizing intermediate memory for long sequences training. Advances in Neural Information Processing Systems (NeurIPS).
[5] Liu, H., Zaharia, M., & Abbeel, P. (2023). Blockwise parallel transformer for large context models. Advances in Neural Information Processing Systems (NeurIPS).

DOI: 10.54254/2755-2721/2026.GU33803

[6] Liu, H., Zaharia, M., & Abbeel, P. (2024). Ring attention with blockwise transformers for near-infinite context. In Proc. 12th International Conference on Learning Representations (ICLR).
[7] Jacobs, S. A., et al. (2023). DeepSpeed Ulysses: System optimizations for enabling training of extreme long sequence transformer models. arXiv preprint arXiv: 2309.14509.
[8] Fang, J., & Zhao, S. (2024). USP: A unified sequence parallelism approach for long context generative AI. arXiv preprint arXiv: 2405.07719.
[9] Brandon, W., et al. (2023). Striped attention: Faster ring attention for causal transformers. arXiv: 2311.09431.
[10] Wang, Z., et al. (2025). WLB-LLM: Workload-balanced 4D parallelism for large language model training. In Proc. 19th USENIX Symposium on Operating Systems Design and Implementation (OSDI).
[11] Jiang, C., et al. (2025). DCP: Addressing input dynamism in long-context training via dynamic context parallelism. In Proc. 31st ACM Symposium on Operating Systems Principles (SOSP).
[12] Yao, J., et al. (2025). Training ultra long context language model with fully pipelined distributed transformer. In Proc. 8th Conference on Machine Learning Systems (MLSys).
[13] NVIDIA Corporation. (2026). NVIDIA A100 Tensor Core GPU Architecture. https: //images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf.
[14] NVIDIA Corporation. (2026). NVIDIA H100 Tensor Core GPU Architecture. https: //resources.nvidia.com/cn-hopper-architecture.
[15] Rabe, M. N., & Staats, C. (2021). Self-attention does not need O(n2) memory. arXiv preprint arXiv: 2112.05682.
[16] Vaswani, A., et al. (2017). Attention is all you need. Advances in Neural Information Processing Systems (NeurIPS), 30.
[17] Yeap, G., et al. (2024). 2nm platform technology featuring energy-efficient nanosheet transistors and interconnects co-optimized with 3DIC for AI, HPC and mobile SoC applications. In IEEE International Electron Devices Meeting (IEDM).
[18] Li, C., et al. (2025). H2-LLM: Hardware-dataflow co-exploration for heterogeneous hybrid-bonding-based low-batch LLM inference. In Proc. 52nd Annual International Symposium on Computer Architecture (ISCA).
[19] Zhou, M., Xu, W., Kang, J., & Rosing, T. (2022). TransPIM: A memory-based acceleration via software-hardware co-design for transformer. In Proc. IEEE International Symposium on High-Performance Computer Architecture (HPCA), 1071-1085.
[20] Khani, M., et al. (2021). SiP-ML: High-bandwidth optical network interconnects for machine learning training. In Proc. ACM SIGCOMM Conference, 657-675.