SeCOv2: Highly Memory Efficient Training for LLMs with Million-Token Contexts

Anonymous Author(s)
Affiliation
Address
email

Abstract

Training Large Language Models (LLMs) on long contexts is severely constrained by the prohibitive GPU memory cost of activations. This bottleneck arises because activation memory scales linearly with sequence length, rendering existing frameworks impractical for million-token scales. To overcome this, we introduce SeCOv2, a block-recurrent framework that processes sequences in chunks to maintain a constant activation memory footprint, irrespective of sequence length. We then address the subsequent bottleneck of the linearly growing KV cache with a suite of highly efficient memory management techniques, centered on paged memory allocation and asynchronous CPU offloading. The synergy of these techniques yields a remarkably low memory overhead, adding only 70-160 MB per 10K additional context tokens for 7-8B models. This efficiency enables training a Qwen2.5-7B model with a 4-million-token context on a single 96GB GPU, making a substantial progress over existing open source frameworks that often require large scale clusters for similar tasks. To further enhance time efficiency, SeCOv2 is designed to integrate seamlessly with tensor parallelism and sparse attention, accelerating the overall training process. We release our code at anonymous GitHub.

1 Introduction

Long-context modeling remains a formidable challenge in the training of Large Language Models (LLMs), a problem rooted not in training time, but in the prohibitive GPU memory overhead associated with long sequences [26, 36]. Research indicates that significant long-context capabilities can be achieved with a surprisingly small number of training steps (as few as one thousand) [28, 9] and that reaching a commercial standard may only require fine-tuning on approximately 5% of the original pretraining token count [13, 3].

Despite this modest data requirement, the number of tokens processed in a single training iteration grows dramatically, causing GPU memory consumption to scale linearly with context length and rapidly exhaust available resources. For instance, a model with a 4× Grouped Query Attention (GQA) [1] ratio requires 64GB of HBM for the KV cache alone when processing a 256K context, a demand that single-handedly overwhelms a 96GB H20 GPU. This figure does not even account for the substantial memory footprint of other network activations, which often makes 128K tokens a practical ceiling for many models [14, 38]. The immense memory pressure from the KV cache and activations dwarfs the optimizer state, rendering popular memory-saving techniques like ZeRO3 offloading [31] and tensor parallel [32] insufficient to overcome this fundamental bottleneck.

While various training-free methods for long-context extension have been proposed [18, 16, 2], their practical utility is often overstated. As established in prior work [12], common evaluation metrics such as Perplexity (PPL) and targeted retrieval tasks like Needle-in-a-Haystack (NIAH) are

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

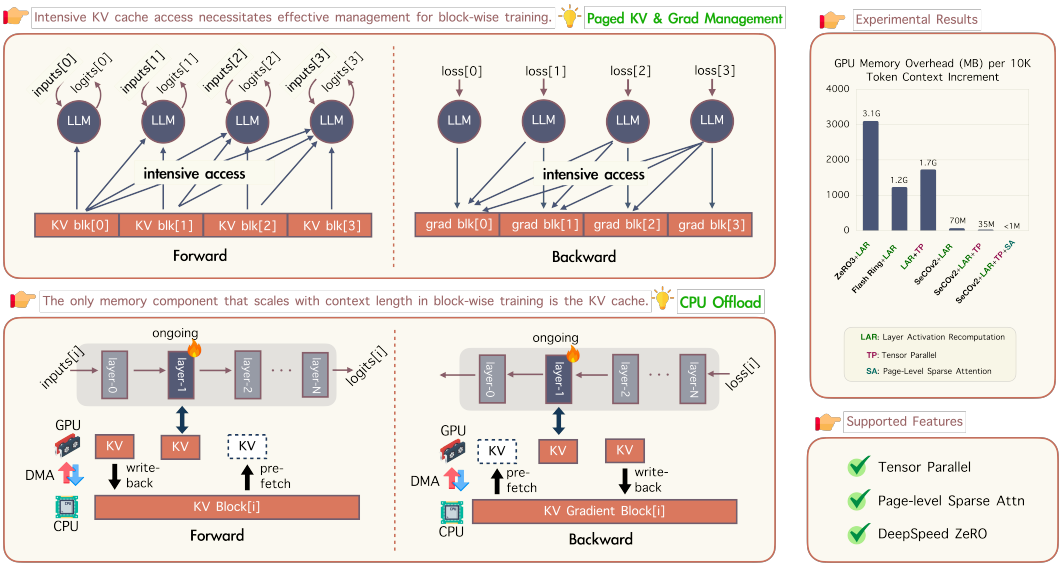


Figure 1: **Overview of the SeCOv2 Training Framework.** (*Top*) SeCOv2 processes long sequences in blocks, maintaining a constant activation memory footprint regardless of context length. (*Bottom*) This exposes the KV cache as the primary memory bottleneck in block-wise training. To solve this, we offload the cache to CPU memory and apply sparse attention, achieving near-constant GPU memory usage. (*Right*) The combined approach yields a >1000× reduction in memory consumption for every additional 10K context compared to strong baselines.

unreliable indicators of true long-range reasoning abilities. Indeed, [12, 3] emphasizes that achieving commercial-grade performance requires dedicated contiguous pretraining.

To this end, this paper directly confronts the central problem: *how to train longer-context models with fewer resources, without compromising performance?* We introduce a block-recurrent training framework [21] that processes sequences in segments, analogous to an RNN. During the forward pass, each block's activations are computed and then immediately discarded. When needed for the backward pass, they are recomputed on-the-fly. This strategy maintains a constant activation memory footprint equivalent to a single block, regardless of the total sequence length.

However, a critical distinction from standard RNNs complicates this picture: Transformer blocks are globally coupled via the KV cache. Processing any single block necessitates access to the KV caches from all preceding blocks. This global dependency introduces two primary bottlenecks: (1) *Memory Bottleneck*: The KV cache and its associated gradients must be retained throughout the training step, scaling linearly with the context length. (2) *Computational Overhead*: The forward and backward passes require frequent access to the entire KV cache and its gradient, introducing significant latency.

To overcome these challenges, we have engineered a highly integrated system centered on efficient KV cache management, synergistically combines several co-designed optimizations:

- *Paged KV Cache and Gradient Management.* We introduce a paged memory manager (inspired by [19]) for both the KV cache and its gradients. This design eliminates expensive `concatenation`, `reallocation` and `copy` operations and mitigates memory fragmentation when appending new key-value pairs.
- *Specialized Op and Kernels.* We execute all operations on the KV cache and its gradients directly within the Op, making them opaque to PyTorch's autograd system. This avoids storing the KV cache as an activation and allows for direct gradient accumulation via CUDA `atomic_add`, improving both storage and computational efficiency.
- *Asynchronous KV Cache CPU Offloading.* Since the KV cache and its gradients are the only components whose memory grows with context length, we designed an asynchronous mechanism to preemptively offload them to CPU memory. Experiments confirm that the resulting data transfer latency is effectively masked by the attention computation.

Table 1: **Comparison of Long-Context Training Methods.** SeCO series uniquely achieve $\mathcal{O}(1)$ activation memory complexity, where N denotes the context length. This key advantage allows SeCOv2, when combined with other techniques, to theoretically enable single-GPU training on contexts exceeding 10M tokens.

Method	GPUs	Act Memory	Exact Attn	Max Ctx Len.
ZeRO3 Offload [31]	8×H20	$\mathcal{O}(N)$	✔	64K
Ring Flash Attention [25]	8×H20	$\mathcal{O}(N/8)$	✔	256K
Tensor Parallelism [17, 32]	8×H20	$\mathcal{O}(N/8)$	✔	256K
SeCO [21]	1×H20	$\mathcal{O}(1)$	✔	128K
SeCOv2	1×H20	$\mathcal{O}(1)$	✔	4M
SeCOv2 + Sparse Attn	1×H20	$\mathcal{O}(1)$	✗	10M+

- *Page-level Sparse Attention.* Our paged memory architecture natively supports page-level sparse attention [40, 27]. This reduces the computational complexity of attention from quadratic to nearly linear, significantly accelerating training. Crucially, it also minimizes communication overhead from our offloading mechanism, as only the sparsely attended-to pages must be transferred from CPU memory back to the GPU for computation.

The synergy of these techniques yields exceptional GPU memory efficiency. Our empirical results show that for every additional 10K tokens of context, the end-to-end training memory overhead increases by a mere 160MB for LLaMA3-8B [14] and 70MB for Qwen2.5-7B [38].

While context parallelism (CP) methods [25, 23, 32] are also highly memory efficient, they do not extend the maximum training sequence length on a single GPU. Consequently, training a Qwen2.5-7B [38] model with a 4M context length requires a large cluster of 256×H100 [37]. In contrast, our approach enables training Qwen2.5-7B [38] with 4M-token context on a single 96GB GPU, representing a substantial advance in resource efficiency.

2 Related Work

Efficient Long-Context LLM Training. A core challenge in extending language models to long contexts is their failure to generalize to token distances and positional encodings not seen during pretraining [16]. While continued training can resolve these issues, the associated GPU memory costs are often prohibitive. This has motivated the development of training-free context extension methods, which typically modify positional encodings to accommodate longer sequences [6, 16, 18, 2]. These approaches, however, introduce significant trade-offs. Methods based on interpolation or extrapolation of positional encodings [6] often degrade performance on short-text tasks by reducing the resolution of positional information [12]. Others that reuse position indices [16, 18, 2] achieve high scores on perplexity benchmarks but have been shown to fail on tasks requiring deep contextual understanding [12], indicating a superficial grasp of the extended context.

Given these limitations, state-of-the-art commercial and open-source models [14, 38, 24] still rely on fine-tuning to expand their context windows. However, the prevailing trend has shifted from algorithmic efficiency [7] towards overcoming memory barriers with massive computational resources [25, 20, 17, 32]. For instance, recent efforts [37] have required 256×H100 GPU cluster to extend LLaMA3.1 [14] from 128K to 4M context.

Serial vs. Parallel Training Paradigms. Language model training architectures can be broadly classified as parallel (Transformers [35], CNNs [29, 11], SSMs [10, 15]) or serial (RNNs [34]). Parallel training processes an entire sequence in a single forward and backward pass, maximizing GPU utilization and enabling scalability. However, this paradigm's memory footprint, which scales linearly with sequence length, creates a significant bottleneck for long-context models. In contrast, serial training is highly memory efficient, as it only activates a small part of the network at any given time, but at the cost of reduced parallelism.

For long-context fine-tuning where dataset sizes are often moderate, memory consumption, not raw throughput, is the primary constraint [28, 12]. This motivates a hybrid approach that processes

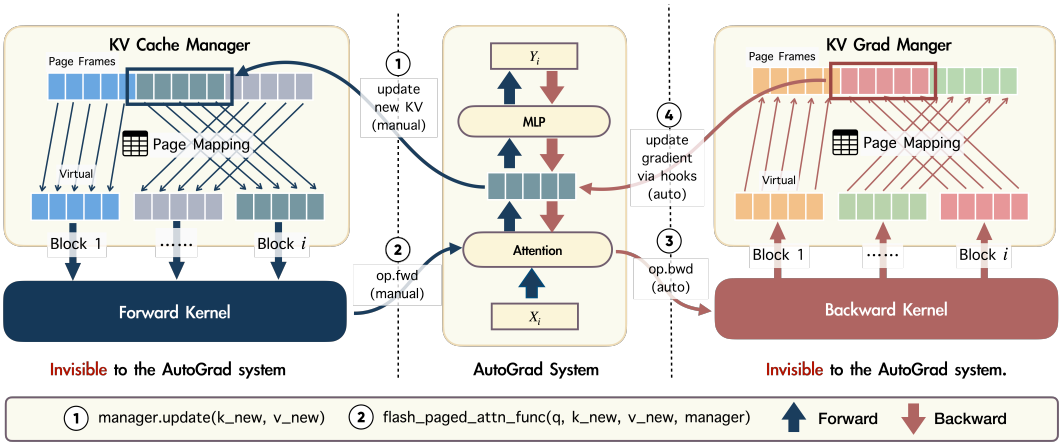


Figure 2: **An Overview of the Efficient KV Cache Management Framework.** Our framework optimizes frequent KV cache operations via two key strategies: (1) We bypass the autograd engine and manually manage KV cache values and their gradients. This ensures the portion of the computation graph handled by autograd has a constant memory footprint, resulting in $\mathcal{O}(1)$ activation memory. (2) We employ a paged memory scheme for the cache, which improves efficiency, reduces fragmentation, and natively supports page-level sparse attention [40, 27].

sequences in blocks—parallel within each block and serial between them. This block-wise strategy is already standard in SOTA LLM inference engines (vLLM [19], FlashInfer [39]), where it incurs negligible latency. However, its application to training remains underdeveloped. An early exploration, SeCO [21], introduced block-wise training but lacked critical optimizations at the operator and memory management levels. As a result, it could only train a 16K context model on a single RTX 3090 GPU. Under identical conditions, our fully-optimized system extends this capability by more than $10\times$, demonstrating a dramatic improvement in efficiency.

Context Parallelism. Context Parallelism (CP) and our proposed method both segment long sequences but diverge significantly in their scalability and overhead profiles.

CP, inspired by Ring Attention [25] and implemented in frameworks like Megatron-LM [32], DeepSpeed Ulysses [17] and TorchTitan [22], distributes a single sequence across multiple GPUs. This design introduces substantial inter-GPU communication overhead that scales quadratically with GPU count. In contrast, our serial approach's overhead is limited to repeated KV cache accesses and the latency from CPU offloading.

The trade-off is clear: CP excels with large GPU clusters but requires a high GPU count (e.g., at least $128\times$80GB GPUs for a 4M context) and specialized, high-bandwidth networks to manage its communication demands. Our method, however, maintains moderate and predictable overhead, eliminating the need for large-scale GPU clusters and specialized interconnects.

3 Preliminary: Foundations of Block-Recurrent Training

The core strategy of SeCOv2 is rooted in the principles of sequential processing, which have long been used to make training Recurrent Neural Networks (RNNs) memory efficient. We first revisit this foundation and then describe how it can be adapted to the Transformer architecture, setting the stage for the challenges that SeCOv2 is designed to solve.

Activation Recomputation in Sequential Models. In a standard RNN, the forward pass iteratively computes an output y_i and a new hidden state m_i at each timestep i:

$$y_i, m_i, \mathbf{a_i} \leftarrow \mathcal{F}_{\text{RNN}}(x_i, m_{i-1}; \Theta) \tag{1}$$

where $\mathbf{a_i}$ represents the intermediate activations cached for the backward pass. For long sequences, storing all activations $\{\mathbf{a}_1, ..., \mathbf{a}_T\}$ creates a memory bottleneck that scales linearly with sequence length. *Activation recomputation* [33, 4] circumvents this by discarding $\mathbf{a_i}$ during the forward pass and regenerating it on-the-fly just before its use in the backward pass. This trades a modest amount of computation for a significant reduction in memory.

Adapting Sequential Processing to Transformers. While Transformers are inherently parallel, they can be trained using a similar block-wise, sequential paradigm. We partition a long input sequence into S blocks, $\{X_1, X_2, ..., X_S\}$, and process them serially. The forward pass for block i attends to the KV caches from all preceding blocks, which act analogously to an RNN's hidden state:

$$Y_i, M_i, \mathbf{A_i} = \mathcal{F}_{\text{Transformer}}(X_i, \{M_1, M_2, ..., M_{i-1}\}; \Theta) \tag{2}$$

Here, M_i is the KV cache generated by block i, and $\mathbf{A_i}$ represents its intermediate activations. By applying the same activation recomputation strategy, we discard $\mathbf{A_i}$ after the forward step and recompute it for the backward pass:

$$\mathrm{d}\Theta, \{\mathrm{d}M_1, ..., \mathrm{d}M_{i-1}\} \leftarrow \text{backward}(\mathrm{d}Y_i, \mathrm{d}M_i, \mathbf{A_i}) \tag{3}$$

This block-recurrent approach ensures that the activation memory footprint remains constant, determined only by the size of a single block. Consequently, this paradigm fundamentally shifts the memory bottleneck: *the challenge is no longer the activations, but the management of the KV cache* $\{M_1, ..., M_S\}$ *and its gradients, which still grow linearly with the sequence length.*

4 Methodology: The SeCOv2 Framework

The block-recurrent paradigm effectively solves the activation memory bottleneck but, as established in Section 3, shifts the primary challenge to managing the linearly growing KV cache, which is a non-trivial issue where naive implementations can inflate peak memory usage by up to 3× due to fragmentation and reallocation overhead (Figure 3). SeCOv2 addresses this challenge through a highly-integrated, systems-level approach. Our methodology is built on three synergistic pillars: (1) an efficient, paged memory system for on-GPU cache management, (2) an asynchronous offloading mechanism to scale beyond the limits of GPU memory, and (3) integration with sparse attention to reduce the computational burden.

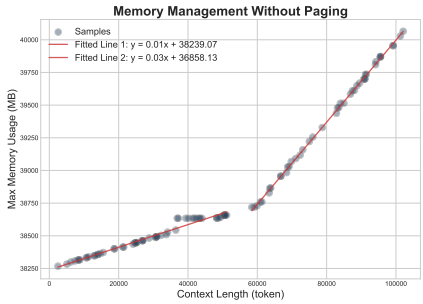


Figure 3: With a naive implementation [21], peak memory allocation for the long-context KV cache can be 3× higher than anticipated.

4.1 Efficient On-GPU KV Cache Management

In our block-wise regime, the KV cache grows dynamically, rendering standard memory management that relies on contiguous tensors highly inefficient. Such approaches lead to expensive `concatenate` and `reallocate` operations and suffer from severe memory fragmentation. To solve this, we introduce the following optimizations.

Paged Memory for Cache and Gradients. We implement a paged memory manager, inspired by vLLM [19], for both the KV cache and its corresponding gradients. As shown in Figure 2, tensors are partitioned into smaller, fixed-size, non-contiguous memory pages stored in a pre-allocated pool. This design completely eliminates reallocation overhead and fragmentation. A potential drawback, reduced L2 cache hit rates, is mitigated by an allocator policy that preferentially places pages from the same logical block into contiguous physical frames.

Operator-Level Optimizations. To maximize throughput, our paged system is coupled with deep operator-level optimizations. We bypass PyTorch's autograd system for KV cache gradient updates; our custom CUDA kernels use a direct `atomic_add` operation to accumulate gradients in place, avoiding the overhead of intermediate buffers and redundant read/write cycles. Furthermore, our recomputation pipeline is optimized to skip the unnecessary recalculation of the KV cache, which is already preserved from the forward pass, further reducing wasteful computation.

4.2 Asynchronous CPU Offloading for Extreme Scalability

While paged management optimizes on-GPU memory, the KV cache's size is ultimately bounded by the GPU's HBM. To transcend this physical limitation, we introduce an asynchronous CPU offloading and pre-fetching mechanism. Leveraging the layer-by-layer execution of Transformers, we ensure only the KV cache for the currently active layer needs to reside in GPU memory. As depicted in Figure 1 (Bottom), while the GPU processes layer i, dedicated CUDA streams concurrently pre-fetch the cache for layer $i+1$ from CPU memory while offloading the already-used cache for layer $i-1$. When the block size is sufficiently large, the data transfer latency is effectively masked by the layer's computation, resulting in a near-constant GPU memory footprint for the KV cache across any sequence length (Table 2).

Table 2: Our CPU offload mechanism ensures the communication overhead is fully masked by computation.

Context	KV Size (GB)	Latency (s)		Throughput (GB/s)	
		flash attn	pre-fetch	flash attn	pre-fetch
100K	0.4	0.0324	0.0172	12.32	23.25
1M	4.0	0.3285	0.1668	12.17	24.17

4.3 Accelerating Training via Sparse Attention

With memory constraints resolved, training time becomes the final frontier, driven by both the $\mathcal{O}(N^2)$ computational complexity of attention and the communication overhead of offloading a growing KV cache. Our paged memory architecture provides an elegant solution to both issues through its native compatibility with page-level sparse attention techniques like Native Sparse Attention (NSA) [40].

This integration yields two critical benefits. First, it reduces the computational complexity to be nearly linear with respect to sequence length, making the attention calculation itself feasible at the million-token scale. Second, it synergizes perfectly with our KV cache offloading strategy. Instead of pre-fetching an ever-growing number of pages from the CPU for each block, sparse attention requires access to only a fixed-size subset of the KV cache. This property ensures that the data payload for each pre-fetch and offload operation remains constant, irrespective of the total sequence length. This dramatically reduces the CPU-GPU communication load, preventing it from becoming a new bottleneck at extreme scales and making training on ultra-long sequences truly feasible.

5 Experiments

In this section, we present a comprehensive empirical evaluation of SeCOv2. Our experiments are designed to validate three core claims: (1) our system is numerically correct and does not compromise model accuracy; (2) it achieves state-of-the-art memory efficiency and scalability on ultra-long sequences; and (3) it outperforms existing long-context training methodologies in resource efficiency.

5.1 Experimental Setup

Unless otherwise specified, all experiments adhere to the following configuration.

Model and Dataset. We use the Qwen2.5-7B model [38], a powerful foundation model, for all efficiency and scalability benchmarks. The training data is sourced from the PG19 dataset [30]. To construct sequences of the required lengths for our long-context experiments, we duplicate and concatenate samples from the dataset.

Training Configuration. All experiments are conducted with full-parameter fine-tuning using `bfloat16` precision and the AdamW optimizer. Since our primary focus is on system efficiency (i.e., time and memory), we keep all other model and training hyperparameters at their default values to ensure a fair comparison of the underlying systems. The per-GPU batch size is set to 1 to isolate the impact of sequence length on performance.

Baseline Methods. Our primary baseline for standard parallel training utilizes a GQA-compatible implementation of FlashAttention-2 (v2.5.8) [8], which represents a highly optimized industry standard. For comparisons involving distributed training, we benchmark against DeepSpeed ZeRO3 [31] and Ring Flash Attention (RFA) [23], a state-of-the-art implementation of Ring Attention [25].

SeCOv2 Configuration. For all SeCOv2 experiments, we use a consistent block size of 4096 tokens and a memory page size of 128 tokens, which we found to offer a robust balance of performance and memory overhead. For experiments involving sparse attention, we set the sparse page budget to 128 pages, corresponding to attending to 16K tokens' worth of KV cache per query block.

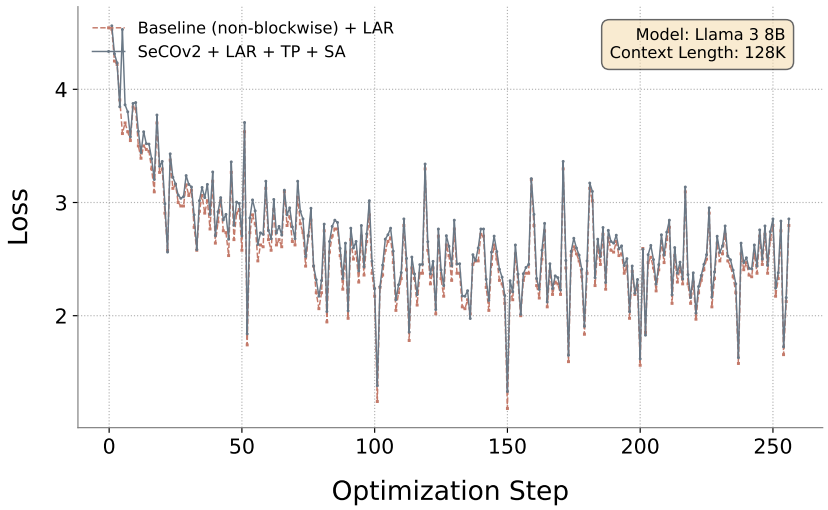


Figure 4: **Accuracy Validation.** Our SeCOv2 implementation, configured with 12.5% sparse attention and 4-GPU tensor parallelism, achieves a final training loss nearly identical to that of a standard baseline using FlashAttention-2 (v2.5.8). This result confirms the numerical correctness and reliability of our system, showing that our optimizations do not compromise model convergence.

Metrics and Measurement. We measure peak GPU memory usage per device using PyTorch's `max_memory_allocated()` function. Per-iteration latency is measured precisely using CUDA events. To ensure statistical reliability and minimize noise, all reported results are the minimum values obtained from 3 independent runs. We explicitly denote when acceleration techniques such as tensor parallelism (TP), layer activation recomputation (LAR), or sparse attention (SA) are applied.

5.2 System Correctness and Accuracy Validation

A primary concern with any novel training framework that introduces custom kernels and unconventional dataflow patterns is its potential impact on model convergence. To verify that SeCOv2's optimizations (specifically its block-wise processing, custom sparse attention, and tensor parallelism) do not introduce numerical discrepancies, we conducted an accuracy validation experiment.

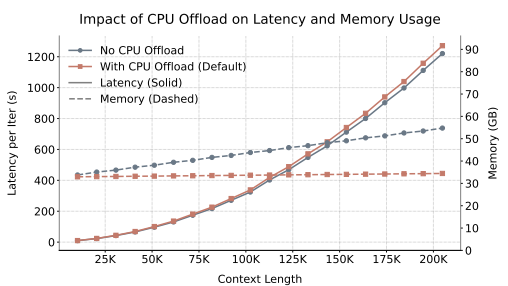
We trained the model for 256 steps and compared the training loss curve of SeCOv2 against the FlashAttention-2 baseline. For this test, SeCOv2 was configured with a 12.5% sparsity level and 4-way tensor parallelism. As illustrated in Figure 4, the loss curve of SeCOv2 closely tracks that of the baseline. The final error difference after 256 steps is negligible (approximately 0.03), confirming that our system is correctly implemented and maintains the convergence properties of standard training.

5.3 Core Performance and Ablation Studies

Having validated the correctness of our system, we now analyze its core performance, scalability, and the impact of its key architectural components.

Efficiency and Scalability up to 4M Tokens. A key goal of SeCOv2 is to unlock training on sequence lengths previously considered infeasible on single-GPU setups. We benchmarked the per-iteration training time and memory of SeCOv2 on contexts ranging from 2K to an unprecedented 4M tokens. Figure 5 showcases the results for both full-attention and sparse-attention variants. SeCOv2 demonstrates remarkable memory efficiency in both configurations, with memory growth being exceptionally slow. When augmented with sparse attention, the training time exhibits a much more favorable, near-linear growth trajectory, making ultra-long context training practically achievable.

Ablation on CPU Offload. The central mechanism for achieving this memory efficiency is our asynchronous KV cache offloading. To quantify its impact, we ablated the CPU offload feature.



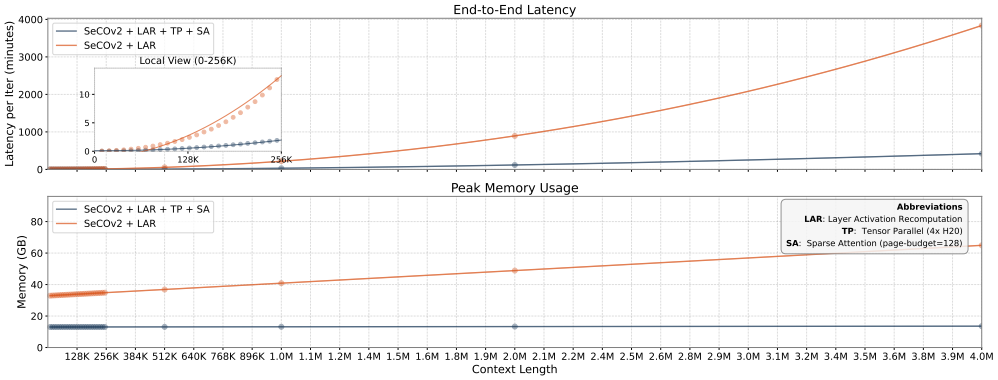


Figure 5: **Scalability up to 4M Tokens.** SeCOv2 demonstrates exceptional memory efficiency for sequence lengths up to 4 million tokens. With full attention (blue), memory usage grows slowly. With sparse attention (red), memory is nearly constant, and training time scales more favorably.

Figure 6 plots the system latency and memory footprint with and without offloading. The results are stark: enabling offloading introduces a minor latency overhead of approximately 5%, which is primarily the cost of initiating data transfers. In exchange, the GPU memory footprint is dramatically reduced and remains almost constant even as the context grows to 200K tokens. This finding validates two critical points: (1) the KV cache is indeed the dominant memory bottleneck in block-recurrent training, and (2) our asynchronous offloading strategy effectively mitigates this bottleneck by hiding data transfer latency behind computation.

Ablation on Block Size. The block size in SeCOv2 is a crucial hyperparameter that trades off between computational parallelism and memory for activations. We ablated the block size, doubling it from 512 to 4096. As shown in Figure 7, increasing the block size improves computational efficiency (reduces training time) due to better hardware utilization from larger matrix multiplications. However, these gains exhibit diminishing returns, with performance saturating beyond a block size of 4096. Meanwhile, memory consumption shows minimal variation. This analysis justifies our choice of 4096 as a default, as it hits a sweet spot for performance. It is also worth noting the theoretical limit: as the block size approaches infinity, our method's performance degenerates to that of parallel training.

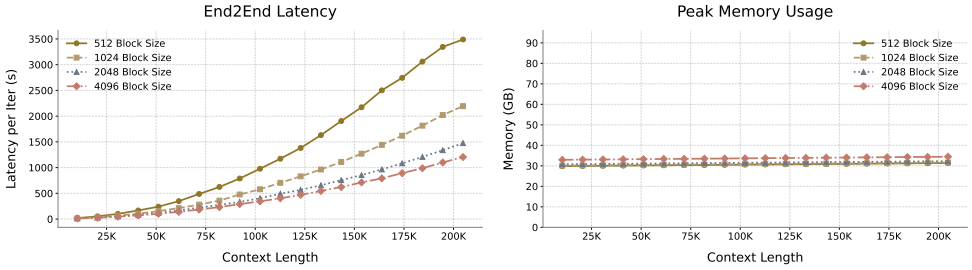


Figure 7: Increasing the block size from 512 to 4096 yields diminishing marginal returns in computational efficiency, with performance eventually converging to the parallel training baseline (which corresponds to an ∞ block size). Memory consumption remains stable across block sizes.

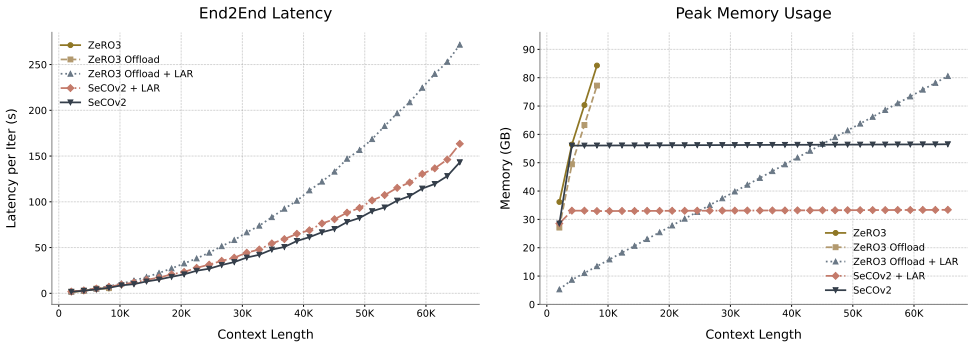


Figure 8: Compared to the widely-recognized DeepSpeed ZeRO3 [31], SeCOv2 demonstrates substantially higher per-GPU efficiency with respect to both computation time and memory usage, even when ZeRO3 is run on a multi-GPU system.

5.4 Comparison with State-of-the-Art Methods

Finally, we benchmark SeCOv2 against leading memory-saving and long-context training techniques to position its performance within the current landscape.

Comparison with ZeRO3-Offload. Both SeCOv2 and DeepSpeed's ZeRO3-Offload [31] leverage CPU offloading to conserve GPU memory. We conducted a direct comparison to evaluate their relative merits. We benchmarked three ZeRO3 configurations on an 8xH20 GPU cluster (per-device batch size 1): (1) pure ZeRO3, (2) ZeRO3 with parameter offloading, and (3) ZeRO3 with both parameter and LAR. SeCOv2 was run on a single H20 GPU. The results, reported for GPU 0 in Figure 8, clearly show that SeCOv2 offers substantially higher per-GPU efficiency. It achieves lower latency and consumes significantly less memory, highlighting the effectiveness of targeting the KV cache bottleneck directly over ZeRO's more general approach to offloading optimizer states, parameters, and activations.

Table 3: Throughput comparison with context parallel methods. We compared the per-GPU training throughput for 128K and 256K context lengths. Results shows that the SOTA implementation, Ring Flash Attention (RFA) [23], achieves a throughput comparable to that of SeCOv2. (The results highlighted in red are from [5].)

Context	**Ring Attention** [25]		**RFA** [23]	**SeCOv2+TP**	
	8×A100	16×TPUv4	8×H20	1×H20	4×H20
128K	-	-	232 tok/s	240 tok/s	229 tok/s
256K	50 tok/s	49 tok/s	125 tok/s	139 tok/s	121 tok/s

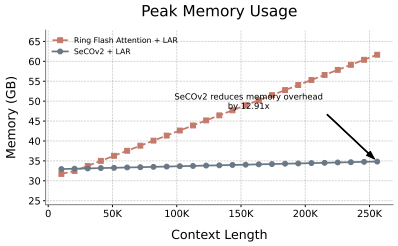


Figure 9: Memory usage compared to context parallel methods.

Comparison with Context Parallelism Methods. Context Parallelism (CP) methods, such as Ring Attention [25], also partition input sequences to manage memory. We benchmarked SeCOv2 against Ring Flash Attention (RFA) [23], a highly optimized and representative open-source CP implementation. Both methods were evaluated for per-device training throughput and memory usage on a multi-GPU setup. The results presented in Table 3 and Figure 9 demonstrate that SeCOv2 delivers throughput comparable to the heavily optimized RFA. However, SeCOv2 achieves this with substantially improved memory efficiency, consuming significantly less GPU memory. This makes SeCOv2 a more resource-efficient option, enabling the training of longer-context models on smaller or less numerous GPUs.

6 Conclusion and Limitations

Conclusion. In this work, we introduced SeCOv2, a highly memory efficient training framework that significantly lowers the resource barrier for training LLMs on million-token contexts. By combining

a block-recurrent training strategy with on-the-fly activation recomputation, we achieve a constant memory footprint for activations, irrespective of sequence length. We address the subsequent KV cache bottleneck through a suite of co-designed optimizations: a paged memory manager for the cache and its gradients, autograd-decoupled cache operations, and an asynchronous CPU offloading mechanism. The synergy of these techniques enables training a 4-million-token context Qwen2.5-7B model on a single 96GB GPU, a task that previously required large-scale clusters. Our framework is designed to be a plug-and-play wrapper, compatible with existing training systems, to help democratize research and development in ultra-long-context LLMs.

Limitations. The primary limitation of SeCOv2 is the trade-off between memory efficiency and computational time. The serial, block-wise processing and activation recomputation introduce latency overhead compared to standard parallel training. While we demonstrate that this overhead is manageable and can be partially mitigated by larger block sizes and integration with tensor parallelism or sparse attention, it remains an inherent characteristic of the design. Furthermore, our asynchronous CPU offloading mechanism relies on the computational workload of a block being sufficient to hide data transfer latency, which may not hold for smaller models or block sizes. Finally, this work focuses on the systems-level efficiency of the training framework itself; we do not propose new long-context learning recipes or data curation strategies, which are orthogonal but essential for achieving optimal model performance.

References

[1] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebron, et al. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *EMNLP*, 2023.

[2] C. An, F. Huang, J. Zhang, S. Gong, X. Qiu, et al. Training-free long-context scaling of large language models. In *ICML*, 2024.

[3] Y. Bai, X. Lv, J. Zhang, Y. He, J. Qi, L. Hou, et al. LongAlign: A recipe for long context alignment of large language models. In *EMNLP*, 2024.

[4] W. Bencheikh, J. Finkbeiner, and E. Neftci. Optimal gradient checkpointing for sparse and recurrent architectures using off-chip memory. *arXiv preprint arXiv:2412.11810*, 2024.

[5] W. Brandon, A. Nrusimha, K. Qian, Z. Ankner, T. Jin, et al. Striped attention: Faster ring attention for causal transformers. 2023.

[6] S. Chen, S. Wong, L. Chen, and Y. Tian. Extending context window of large language models via positional interpolation. *arXiv preprint arXiv:2306.15595*, 2023.

[7] Y. Chen, S. Qian, H. Tang, X. Lai, Z. Liu, et al. Longlora: Efficient fine-tuning of long-context large language models. In *ICLR*, 2024.

[8] T. Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *ICLR*, 2024.

[9] Y. Ding, L. L. Zhang, C. Zhang, Y. Xu, N. Shang, et al. Longrope: extending llm context window beyond 2 million tokens. In *ICML*, 2024.

[10] D. Y. Fu, T. Dao, K. K. Saab, A. W. Thomas, A. Rudra, et al. Hungry hungry hippos: Towards language modeling with state space models. In *ICLR*, 2023.

[11] D. Y. Fu, E. L. Epstein, E. Nguyen, A. W. Thomas, M. Zhang, T. Dao, A. Rudra, and C. Re. Simple hardware-efficient long convolutions for sequence modeling. In *ICLR*, 2023.

[12] T. Gao, A. Wettig, H. Yen, and D. Chen. How to train long-context language models (effectively). *arXiv preprint arXiv:2410.02660*, 2024.

[13] T. Gao, A. Wettig, H. Yen, and D. Chen. How to train long-context language models (effectively). In *ICLR*, 2025.

[14] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[15] A. Gu, K. Goel, and C. Ré. Efficiently modeling long sequences with structured state spaces. In *ICLR*, 2022.

[16] C. Han, Q. Wang, H. Peng, W. Xiong, Y. Chen, et al. Lm-infinite: Zero-shot extreme length generalization for large language models. In *NAACL*, 2024.

[17] S. A. Jacobs, M. Tanaka, C. Zhang, M. Zhang, R. Y. Aminadabi, et al. Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models. In *PODC*, 2024.

[18] H. Jin, X. Han, J. Yang, Z. Jiang, Z. Liu, et al. Llm maybe longlm: Selfextend llm context window without tuning. In *ICML*, 2024.

[19] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with pagedattention. In *SOSP*, 2023.

[20] D. Li, R. Shao, A. Xie, E. P. Xing, X. Ma, et al. DISTFLASHATTN: Distributed memory-efficient attention for long-context LLMs training. In *CoLM*, 2024.

[21] W. Li, Y. Zhang, G. Luo, D. Yu, and R. Ji. Training long-context llms efficiently via chunk-wise optimization. In *ACL*, 2025.

[22] W. Liang, T. Liu, L. Wright, W. Constable, A. Gu, et al. Torchtitan: One-stop pytorch native solution for production ready llm pre-training. *arXiv preprint arXiv:2410.06511*, 2025.

[23] Z. Lin. Ring flash attention: Ring attention implementation with flash attention. `https://github.com/zhuzilin/ring-flash-attention`, 2025.

[24] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2025.

[25] H. Liu, M. Zaharia, and P. Abbeel. Ringattention with blockwise transformers for near-infinite context. In *ICLR*, 2024.

[26] J. Liu, D. Zhu, Z. Bai, Y. He, H. Liao, H. Que, Z. Wang, C. Zhang, G. Zhang, J. Zhang, et al. A comprehensive survey on long context language modeling. *arXiv preprint arXiv:2503.17407*, 2025.

[27] E. Lu, Z. Jiang, J. Liu, Y. Du, T. Jiang, et al. Moba: Mixture of block attention for long-context llms. *arXiv preprint arXiv:2502.13189*, 2025.

[28] B. Peng, J. Quesnelle, H. Fan, and E. Shippole. YaRN: Efficient context window extension of large language models. In *ICLR*, 2024.

[29] M. Poli, S. Massaroli, E. Nguyen, D. Y. Fu, T. Dao, et al. Hyena hierarchy: towards larger convolutional language models. In *ICML*, 2023.

[30] J. W. Rae, A. Potapenko, S. M. Jayakumar, C. Hillier, and T. P. Lillicrap. Compressive transformers for long-range sequence modelling. In *ICLR*, 2020.

[31] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He. Zero: memory optimizations toward training trillion parameter models. In *SC*, 2020.

[32] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, et al. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2020.

[33] N. S. Sohoni, C. R. Aberger, M. Leszczynski, J. Zhang, and C. Ré. Low-memory neural network training: A technical report. *arXiv preprint arXiv:1904.10631*, 2022.

[34] R. C. Staudemeyer and E. R. Morris. Understanding lstm – a tutorial into long short-term memory recurrent neural networks. *arXiv prepring arXiv:1909.09586*, 2019.

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, et al. Attention is all you need. In *NeurIPS*, 2017.

[36] X. Wang, M. Salmani, P. Omidi, X. Ren, M. Rezagholizadeh, et al. Beyond the limits: a survey of techniques to extend the context length in large language models. In *IJCAI*, 2024.

[37] C. Xu, W. Ping, P. Xu, Z. Liu, B. Wang, et al. From 128k to 4m: Efficient training of ultra-long context large language models. *arXiv preprint arXiv:2504.06214*, 2025.

[38] A. Yang, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2025.

[39] Z. Ye, L. Chen, R. Lai, W. Lin, Y. Zhang, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *arXiv preprint arXiv:2501.01005*, 2025.

[40] J. Yuan, H. Gao, D. Dai, J. Luo, L. Zhao, et al. Native sparse attention: Hardware-aligned and natively trainable sparse attention. 2025.