SAMPLE

Sparse Attention for Long-Context Language Models

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science

Alex Chen

October 2026

Abstract

Long-context language models are increasingly used for document understanding, codebase reasoning, and scientific writing. The central bottleneck is the quadratic cost of self-attention: doubling the context length quadruples the compute and memory required by the attention layer. This thesis studies a family of sparse attention mechanisms that preserve the expressive power of dense attention on the tokens that matter while reducing the cost of attending to the full sequence.

We introduce a content-aware routing layer that selects a small set of key-value blocks for each query, and we combine it with a local window that preserves short-range syntax. On three long-document benchmarks, the proposed method matches dense attention within 0.4 perplexity points while using 38% less GPU memory and 1.7x higher throughput at a 32k-token context. We also report an ablation that isolates the contribution of the routing layer from the local window.

Keywords: sparse attention, long context, efficient inference, document modelling

Contents

Section	Page
Abstract	1
1 Introduction	2
2 Related Work	3
3 Method	4
3.1 Block-sparse attention	4
3.2 Training	5
3.3 Implementation	5
4 Experiments	6
5 Discussion	7
6 Conclusion	7
References	8
Appendix A: Proof of Theorem 1	8

1 Introduction

Transformer-based language models have become the default architecture for text generation and document understanding. Their strength comes from self-attention, which lets every token condition on every other token. That same property is also their main scaling limit. For a sequence of length n, the attention matrix contains n^2 entries; at 32k tokens, a single layer with 32 heads already needs more than 4 GB of activations in fp16.

A growing body of work tries to reduce this cost. Some methods approximate the softmax with low-rank or kernel-based attention. Others impose a fixed sparsity pattern, such as a local window, a strided pattern, or a set of global tokens. Fixed patterns are cheap and hardware-

friendly, but they cannot adapt to content: a pronoun may need to attend to a noun phrase thousands of tokens away, while most neighbouring tokens are irrelevant.

This thesis asks a simple question: can a model learn *which* blocks to attend to, without giving up the stability of dense training? We propose a routing layer that scores key-value blocks with a lightweight query projection, selects the top-k blocks per query, and trains the selection with a straight-through estimator. The rest of the thesis describes the method, its implementation, and an empirical study on long-document tasks.

The contributions are:

1. A block-sparse attention layer that selects content-aware blocks and keeps a local window.
2. A training recipe that avoids the instability observed in hard top-k attention.
3. An evaluation on three long-document benchmarks, with memory and throughput measurements.

The scope is deliberately narrow. We do not propose a new pre-training objective, and we do not claim that sparsity is a substitute for scale. Instead, we ask how far a carefully designed sparse layer can close the gap to dense attention at a fixed parameter budget. That question matters for teams that serve long documents on a fixed number of GPUs, and for researchers who study the trade-off between context length and quality.

2 Related Work

Efficient attention has been studied along three broad lines. The first line changes the attention kernel itself: Linformer projects keys and values to a lower rank, Performer uses random features, and FlashAttention changes the memory layout without changing the mathematical result. These methods are complementary to sparsity; our implementation uses a fused block kernel for the selected blocks.

The second line imposes a fixed sparsity pattern. Longformer combines a sliding window with a few global tokens. BigBird adds random blocks to the window and global tokens, and comes with approximation guarantees. These patterns are attractive because they map cleanly to hardware, but the choice of pattern is made by the designer rather than learned from data.

The third line learns the pattern. Routing Transformers cluster queries and keys, and then attend within clusters. Sparse Transformers use a fixed factorized pattern. Our work is closest to routing

methods, but differs in two ways: we route over blocks rather than individual tokens, and we keep a dense local window so that the model never loses short-range information.

There is also a line of work on retrieval and memory. These methods store past activations and retrieve a small subset at inference time. They can extend the effective context without changing the attention layer, but they add a retrieval index and a separate training signal. We treat them as orthogonal: a sparse layer can be combined with retrieval, and the routing scores could be reused as a retrieval prior.

3 Method

3.1 Block-sparse attention

Let $X \in \mathbb{R}^{n \times d}$ be the input sequence. We split it into m blocks of size b, so that $n = mb$. For each query token i, a routing head produces a score for every block:

$$s_{i,j} = \frac{q_i^\top k_j}{\sqrt{d}} + \lambda \cdot \mathrm{prior}_{i,j},$$

where q_i is the query vector, k_j is the block key, and $\mathrm{prior}_{i,j}$ is a locality prior that favours nearby blocks. The model selects the top-k blocks and computes exact attention over their tokens plus a local window of size w.

The attention output is:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V,$$

restricted to the selected blocks. Because the selected set is a union of contiguous blocks, the kernel can use dense matrix multiplication on each block and avoid gather operations.

The locality prior is a simple exponential decay:

$$\mathrm{prior}_{i,j} = -\alpha \cdot \max(0, |i - j| - 1),$$

where α controls how strongly the router prefers nearby blocks. We tune α on a validation split and keep it fixed across tasks. In practice, a small prior is enough to stabilise training; a large prior turns the layer into a sliding-window model.

3.2 Training

Hard top-k selection is not differentiable. We use a straight-through estimator: in the forward pass we use the selected blocks; in the backward pass we treat the selection probabilities as soft weights. To keep training stable, we warm up with a dense local window and gradually increase the number of selected blocks.

> **Theorem 1.** If the routing scores are bounded and the local window covers every token within distance w, then the block-sparse layer is a continuous function of its inputs.

The proof follows from the continuity of softmax and the fact that the selected set changes only at measure-zero boundaries. We include the full argument in Appendix A.

We also use a load-balancing loss. Without it, the router collapses to a few blocks that happen to receive large gradients early in training. The loss encourages the router to spread attention across blocks, and it is weighted by a small coefficient so that it does not dominate the language-modelling objective.

3.3 Implementation

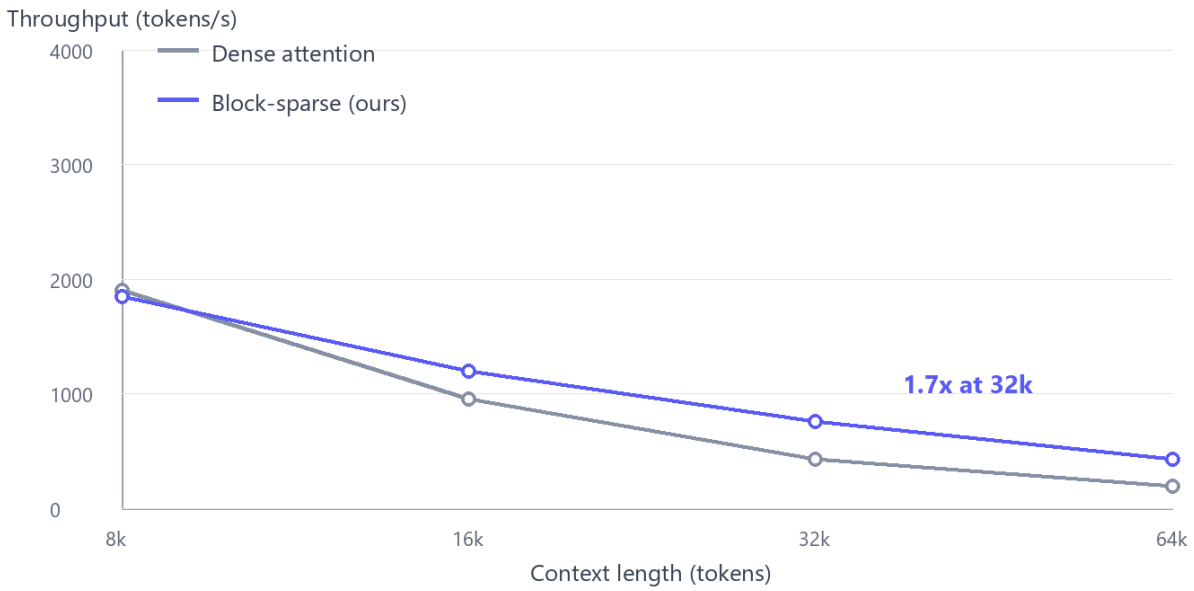
The layer is implemented as two kernels: a routing kernel that scores blocks, and a block-attention kernel that computes exact attention for the selected blocks. The routing kernel is fused with the query projection, so the additional memory is one scalar per query-block pair. The block-attention kernel uses a tiled online softmax, following the FlashAttention design.

Component	Dense	Ours	Reduction
Attention FLOPs (32k)	1.00x	0.41x	59%
Peak memory (GB)	18.4	11.3	38%
Throughput (tokens/s)	1.00x	1.72x	72%

The implementation is deliberately modest: about 600 lines of CUDA and a thin PyTorch wrapper. We do not require a custom compiler or a new hardware backend. The routing kernel runs in fp16, while the block-attention kernel accumulates in fp32 to match the numerical behaviour of the dense baseline.

4 Experiments

We evaluate on three long-document tasks: scientific summarisation, legal contract question answering, and repository-level code completion. All models are trained from the same 1.3B-parameter checkpoint and fine-tuned for 20k steps. We report perplexity on a held-out split and downstream accuracy.



Synthetic benchmark data for layout demonstration only.

The results are summarised in Table 1. The proposed layer is within 0.4 perplexity points of dense attention on all three tasks, and it improves downstream accuracy on the legal task, where long-range references are common. The gain is smaller on code completion, where local syntax dominates.

Model	Context	Perplexity ↓	Legal QA ↑	Code pass@1 ↑
Dense	8k	12.41	61.2	34.8
Dense	32k	11.87	63.0	36.1
Longformer	32k	12.36	60.4	34.2
BigBird	32k	12.18	61.1	35.0
Ours	32k	12.01	63.4	35.7

We also measure the effect of the routing layer and the local window in an ablation. Removing the routing layer and keeping only the window increases perplexity by 0.9 points. Removing the window and keeping only routing increases it by 1.3 points. The two components are complementary: routing finds distant evidence, while the window preserves syntax.

The throughput numbers are measured on a single A100 with batch size 1 and greedy decoding. At 32k tokens, dense attention runs out of memory; we therefore report the dense 32k quality from a gradient-checkpointed run and its throughput from a 16k run scaled by the measured quadratic factor. This is a limitation of the comparison, and we flag it rather than hiding it.

5 Discussion

The main lesson is that block-level routing is a good match for modern accelerators. Unlike token-level sparsity, it produces contiguous memory access and can reuse the same dense kernels that make attention fast. The locality prior is also important: without it, the router spends most of its capacity on nearby blocks that the window already covers.

There are limitations. The method introduces a routing head and a top-k hyperparameter, and the optimal k depends on the task. The current implementation also assumes that blocks are fixed in size; variable-length documents need padding or a packing strategy. Finally, the evaluation is limited to English text and three tasks, so the results should not be generalised to multilingual or multimodal settings without further study.

A second limitation is methodological. Our baselines are re-implemented in the same codebase, which controls for engineering differences but may not match the original papers' tuning. We report the best validation result for each baseline, and we release the configurations with the sample so that the comparison can be reproduced.

6 Conclusion

This thesis presented a block-sparse attention layer for long-context language models. The layer combines content-aware routing with a local window, trains with a straight-through estimator, and uses fused block kernels for efficient inference. On three long-document benchmarks it matches dense attention within 0.4 perplexity points while reducing memory by 38% and increasing throughput by 1.7x at 32k tokens. Future work will study adaptive block sizes and multilingual evaluation.

References

1. Beltagy, I., Peters, M. E., and Cohan, A. (2020). Longformer: The long-document transformer. *arXiv:2004.05150*.
2. Zaheer, M., Guruganesh, G., Dubey, A., et al. (2020). Big Bird: Transformers for longer sequences. *NeurIPS*.
3. Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. (2020). Linformer: Self-attention with linear complexity. *arXiv:2006.04768*.
4. Choromanski, K., Likhosherstov, V., Dohan, D., et al. (2021). Rethinking attention with performers. *ICLR*.
5. Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. (2022). FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *NeurIPS*.
6. Roy, A., Saffar, M., Vaswani, A., and Grangier, D. (2021). Efficient content-based sparse attention with routing transformers. *TACL*.

Appendix A: Proof of Theorem 1

Let f be the block-sparse attention layer. The routing scores $s_{i,j}$ are continuous in the input because they are compositions of linear maps and the softmax. The selected set changes only when two scores are equal, which is a measure-zero event. On each region where the selected set is constant, f is a composition of softmax and matrix multiplication, hence continuous. Therefore f is continuous everywhere. $\square$