

The Theory and Mathematics of Sparse Attention Mechanisms

Timothy M. Fox

October 2026

Abstract

The Transformer architecture, underpinned by the self-attention mechanism, has become the de facto standard for a wide range of sequence processing tasks. However, the standard “dense” attention mechanism has a computational complexity of $O(n^2)$ with respect to the sequence length n , and a naive implementation needs $O(n^2)$ memory as well. This quadratic scaling makes processing long sequences computationally prohibitive. Sparse attention mechanisms have emerged as a powerful solution, approximating the dense attention matrix with a sparse one, thereby reducing the complexity to $O(n\sqrt{n})$, $O(n \log n)$, or even $O(n)$. This document explores the mathematical foundations of sparse attention, detailing its theoretical justification and the common sparsity patterns employed, including a causal local-plus-dilated pattern designed for high-frequency financial data.

1 Introduction

Since its introduction by Vaswani et al. [1], the Transformer architecture has revolutionized sequence modeling across natural language processing, computer vision, and time series analysis. At its core lies the self-attention mechanism, which computes pairwise interactions between all positions in a sequence, enabling the model to capture long-range dependencies without the sequential bottleneck of recurrent architectures.

However, this expressive power comes at a steep computational cost. The standard attention mechanism requires computing an $n \times n$ attention matrix for a sequence of length n , resulting in $O(n^2)$ time complexity and, in a naive implementation, $O(n^2)$ space. For modern applications requiring long context windows—such as document summarization (tens of thousands of tokens), genomic sequence analysis (millions of base pairs), or high-frequency financial data (where the order book feed for a single liquid instrument can produce thousands of updates per second)—this quadratic scaling becomes a fundamental barrier.

Consider a practical example from high-frequency trading: processing a 64K-tick window (65,536 trade ticks, roughly 3–4 hours of trading in a liquid instrument) with dense attention means scoring about 4.3 billion query–key pairs per attention head. A naive implementation materializes this matrix: at 32-bit floating point precision it consumes 16 GB of GPU memory per head, or 64 GB for a modest four-head layer, before accounting for gradients, activations, or model parameters. IO-aware kernels such as FlashAttention [7] avoid storing the matrix by computing the softmax in tiles, which reduces memory to linear in n , but they cannot remove the quadratic arithmetic. The same four-head layer with head dimension 64 still needs about 4.4 trillion floating-point operations per forward pass; even a GPU sustaining one petaFLOP per second would spend over 4 ms on that single layer. For latency-sensitive workloads, the quadratic cost itself, not only its memory footprint, is the barrier.

Sparse attention addresses this challenge by exploiting a key empirical observation: in trained models, each query’s attention tends to concentrate on a small number of keys, and most weights contribute little to the output. By restricting each query to attend only to a carefully selected subset of keys, sparse attention reduces complexity to $O(n \cdot w)$ or $O(n \log n)$ while preserving much of the representational power of dense attention.

This paper provides a rigorous mathematical treatment of sparse attention mechanisms. We formalize the theoretical foundations, analyze common sparsity patterns, and discuss their implications for different domains. While we focus on general principles, we pay particular attention to applications in financial time series, where temporal locality and streaming requirements align naturally with sparse attention architectures.

2 Related Work

The limitations of quadratic-complexity attention have motivated extensive research into efficient alternatives. We review the most influential sparse attention mechanisms that form the foundation of modern long-context Transformers.

Sparse Transformer. Child et al. [2] introduced structured sparsity patterns for attention, demonstrating that carefully designed sparse connectivity could match dense attention performance on generative modeling tasks. Their work proposed factorized attention patterns including strided and fixed attention, reducing complexity from $O(n^2)$ to $O(n\sqrt{n})$.

Longformer. Beltagy et al. [3] combined local sliding window attention with task-specific global attention, enabling processing of documents up to 4,096 tokens. Their key insight was that most tokens benefit from local context while a small number of tokens (e.g., [CLS]) require global information aggregation. This hybrid pattern achieves $O(n \cdot w)$ complexity.

BigBird. Zaheer et al. [4] provided theoretical analysis showing that sparse attention with random, window, and global components can approximate full attention. They proved that such patterns are Turing complete and universal approximators for sequence-to-sequence functions, establishing theoretical foundations for sparse attention mechanisms.

Trainable, hardware-aligned sparsity. More recent work learns which keys to attend to instead of fixing the pattern in advance, and operates on contiguous blocks of keys so that the computation maps onto GPU matrix units. Native Sparse Attention (NSA) [9] combines compressed coarse-grained tokens, selected fine-grained blocks, and a sliding window, and is trained end to end; it matches or exceeds full attention on general and long-context benchmarks while running substantially faster at 64K tokens. MoBA [10] applies a mixture-of-experts style gate to route each query to a few key blocks, can switch between full and sparse attention, and has been deployed in production for long-context serving. These results suggest that sparsity can be learned during pretraining rather than imposed afterward.

Other Notable Work. Additional innovations include Reformer’s locality-sensitive hashing for approximate attention [5], Linformer’s low-rank projections [6], and FlashAttention’s IO-aware algorithm [7], which optimizes dense attention but whose tiling techniques also underpin efficient sparse attention kernels. For streaming inference, Xiao et al. [8] showed that a sliding window alone degrades once the earliest tokens leave the cache, and that keeping a few initial “attention sink” tokens restores stability, a further argument for combining local windows with a small set of persistent positions.

Our presentation synthesizes these ideas, providing a unified mathematical framework for understanding sparse attention mechanisms and their domain-specific applications.

3 Background: The Standard Attention Mechanism

To understand sparse attention, we must first formalize the standard (dense) attention mechanism. Given a sequence of n input token embeddings, we project them into three matrices: Query $Q \in \mathbb{R}^{n \times d_k}$, Key $K \in \mathbb{R}^{n \times d_k}$, and Value $V \in \mathbb{R}^{n \times d_v}$, where d_k is the dimension of the keys and queries and d_v the dimension of the values. We write Q_i , K_j , and V_j for individual rows.

The attention output, in which each token’s representation is a weighted combination of the others’, is computed as:

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

Let’s break this down:

1. **Similarity Scores:** The matrix product $A = QK^\top$ computes the dot product between every query vector Q_i and every key vector K_j . This $n \times n$ matrix, A , represents the unscaled similarity scores.

$$A_{ij} = Q_i \cdot K_j$$

2. **Scaling:** The scores are scaled by dividing by $\sqrt{d_k}$. This is a stabilizing factor to prevent the dot products from growing too large, which could saturate the softmax function and lead to vanishing gradients.

$$\text{Scaled}A_{ij} = \frac{A_{ij}}{\sqrt{d_k}}$$

3. **Softmax Normalization:** The softmax function is applied row-wise to the scaled attention scores, converting them into a probability distribution. The resulting matrix, P , contains weights where P_{ij} is the attention weight from token i to token j .

$$P_{ij} = \frac{\exp(\text{ScaledA}_{ij})}{\sum_{k=1}^n \exp(\text{ScaledA}_{ik})}$$

Note that $\sum_{j=1}^n P_{ij} = 1$ for any given row i .

4. **Weighted Values:** Finally, the attention weights matrix P is multiplied by the Value matrix V to produce the output. Each output vector O_i is a weighted sum of all value vectors, where the weights are the attention probabilities.

$$O_i = \sum_{j=1}^n P_{ij} V_j$$

The critical bottleneck here is the computation of the $n \times n$ attention matrix A (and subsequently P). For a sequence of length 64K, this matrix has over 4 billion elements. Naive implementations also store it; IO-aware kernels avoid the storage but must still compute every entry.

4 The Theory of Sparse Attention

The core hypothesis of sparse attention is that the dense $n \times n$ attention matrix is redundant. Most of the attention scores are small and contribute little to the final output. The information needed to make a prediction for a given token can typically be sourced from a much smaller, localized subset of other tokens.

Sparse attention aims to compute only a subset of the A_{ij} scores, effectively replacing the dense attention matrix P with a sparse matrix P' .

Let S be a set of index pairs (i, j) that we wish to compute attention for. For a query Q_i , its attention is restricted to the set of keys K_j where $j \in S_i = \{j \mid (i, j) \in S\}$.

The sparse attention output for token i is computed by normalizing only over the restricted set S_i :

$$\text{SparseAttention}(Q, K, V)_i = \sum_{j \in S_i} \frac{\exp\left(\frac{Q_i \cdot K_j}{\sqrt{d_k}}\right)}{\sum_{k \in S_i} \exp\left(\frac{Q_i \cdot K_k}{\sqrt{d_k}}\right)} V_j$$

Note that the softmax normalization is computed *only over the attended positions* $j \in S_i$, not over all positions $j \in \{1, \dots, n\}$. This is critical: the denominator sums only over $k \in S_i$, ensuring that attention weights sum to 1 over the sparse connectivity pattern. Because S_i is a set, each key appears in the sum at most once; an implementation that builds S_i as a list of indices from overlapping components must remove duplicates, or the repeated keys will be over-weighted.

In practice, this is often implemented via masking. We set the attention scores for non-attended pairs to $-\infty$ (or, in finite precision, a large negative number) before the softmax operation.

$$\text{ScaledA}'_{ij} = \begin{cases} \frac{Q_i \cdot K_j}{\sqrt{d_k}} & \text{if } j \in S_i \\ -\infty & \text{if } j \notin S_i \end{cases}$$

Since $\exp(-\infty) = 0$, applying softmax to this masked matrix gives $P'_{ij} = 0$ for all $j \notin S_i$ (approximately zero when a finite mask value is used), and the result equals the sparse formula above.

$$P'_{ij} = \frac{\exp(\text{ScaledA}'_{ij})}{\sum_{k=1}^n \exp(\text{ScaledA}'_{ik})}$$

Masking reproduces the sparse output exactly, but on its own it still computes all n^2 scores. The savings come only when the kernel skips masked entries, either by computing the $|S_i|$ scores of each row directly or by skipping tiles of the matrix that are entirely masked.

The key challenge, and where different sparse attention models differ, is in the choice of the sparsity set S . The goal is to choose a set S such that $|S| \in O(n \log n)$ or $O(n)$ while minimizing the approximation error relative to the full attention matrix.

5 Common Sparsity Patterns

The choice of the set S_i for each token i defines the “sparsity pattern.” Several patterns have proven effective.

5.1 Sliding Window (or Local) Attention

For many types of data (text, time series, images), nearby tokens are the most relevant. Sliding window attention formalizes this by allowing each token to attend only to its neighbors within a fixed window radius w .

The set of attended indices for token i is:

$$S_i = \{j \mid |i - j| \leq w, 1 \leq j \leq n\}$$

This is highly efficient, as each token computes at most $2w + 1$ attention scores. The complexity is $O(n \cdot w)$, which is linear in n if w is constant.

5.2 Causal Sliding Window

Autoregressive models, and any model that must run on a live data stream, cannot let a position attend to the future. The causal variant keeps only the current position and the $w - 1$ positions before it:

$$S_i = \{j \mid \max(1, i - w + 1) \leq j \leq i\}$$

Each token computes at most w scores. In a streaming setting, only the last w keys and values need to be kept, so memory per layer is constant in the length of the stream.

5.3 Dilated (or Strided) Sliding Window

A limitation of the simple sliding window is that the receptive field of a single layer is limited: a token can only see information from at most w positions away, and stacking L layers extends this only to $L \cdot w$. To expand the receptive field without increasing computation, the window can be “dilated.”

With an integer dilation factor $d \geq 1$ and a window radius w , the set of attended indices is:

$$S_i = \{j \mid j = i + k \cdot d, k \in \{-w, \dots, -1, 0, 1, \dots, w\}, 1 \leq j \leq n\}$$

This pattern samples positions at intervals of d , allowing the model to see up to $w \cdot d$ positions away with the same cost of at most $2w + 1$ scores per token as a standard sliding window. Setting $d = 1$ recovers the standard sliding window, and the constraint $1 \leq j \leq n$ ensures all indices remain valid within the sequence.

5.4 Block-Sparse Attention

GPUs compute dense matrix tiles far more efficiently than scattered individual entries. Block-sparse attention therefore partitions the sequence into blocks of size b , writes $B(i) = \lceil i/b \rceil$ for the block containing position i , and makes attention decisions per pair of blocks. A simple local version lets each block attend to itself and its neighbors:

$$S_i = \{j \mid |B(i) - B(j)| \leq 1, 1 \leq j \leq n\}$$

Each token attends to at most $3b$ keys, giving $O(n \cdot b)$ complexity, and every computed tile is fully dense, so no work is spent on masked entries inside a tile. Learned methods such as NSA and MoBA keep this block structure but choose the blocks per query instead of fixing them in advance.

5.5 Global Attention

Some tokens in a sequence have broad, summary-level importance (e.g., the [CLS] token in BERT). These tokens should be able to attend to all other tokens, and all other tokens should be able to attend to them.

In this pattern, we pre-select a small number of tokens to have “global” attention. Let G be the set of global token indices.

The full set of attended indices S_i for a token i is a union of its local window and the global tokens:

$$S_i = \{j \mid |i - j| \leq w\} \cup G$$

For a global token $i \in G$, its attention is dense:

$$S_i = \{j \mid 1 \leq j \leq n\}$$

The total cost is $O(n \cdot (w + |G|))$, which remains linear in n as long as the number of global tokens is bounded. Models like the Longformer combine a sliding window with global attention on a few key tokens, achieving a balance between local context and global information integration.

5.6 Fixed Attention

The fixed pattern of Child et al. [2] divides the sequence into blocks of length l and designates the last c positions of every block as summary positions. Each token attends to every position in its own block and to every summary position:

$$S_i = \{j \mid \lceil j/l \rceil = \lceil i/l \rceil\} \cup \{j \mid j \in \{kl - c + 1, \dots, kl\}, 1 \leq k \leq \lceil n/l \rceil\}$$

(intersected with $1 \leq j \leq n$, and with $j \leq i$ for autoregressive models). The summary positions act as relay points: information from any block reaches the rest of the sequence through them in two hops. Each token attends to $l + c\lceil n/l \rceil$ positions, so choosing $l \approx \sqrt{n}$ with c constant gives $O(n\sqrt{n})$ total cost.

5.7 A Financial Pattern: Causal Local Window with Dilated Clusters

Tick data has structure at two time scales. Microstructure effects such as order flow imbalance and short-term volatility depend on the most recent few hundred ticks, while regime context such as the prevailing trend or volatility level is visible in sparser samples reaching much further back. A pattern for this setting combines a causal local window of size w with C dilated clusters of c consecutive ticks, spaced σ ticks apart:

$$L_i = \{j \mid \max(1, i - w + 1) \leq j \leq i\}, \quad D_i = \bigcup_{k=1}^C \{j \mid i - k\sigma - c + 1 \leq j \leq i - k\sigma, j \geq 1\}$$

$$S_i = L_i \cup D_i$$

The pattern is causal, since every $j \leq i$, and each token attends to at most $w + Cc$ positions, so the cost is $O(n \cdot (w + Cc))$. When σ is smaller than w , some clusters overlap the local window; the set union counts those positions once, and an index-list implementation must skip them explicitly to keep the softmax correct. In the `ma-transformer` reference implementation the defaults are $w = 512$, $\sigma = 1000$, $c = 8$, and $C = 10$: each tick attends to at most 592 positions and sees about 10,000 ticks into the past.

6 Conclusion

Sparse attention mechanisms represent a principled approach to scaling Transformer architectures beyond the $O(n^2)$ bottleneck of dense attention. By carefully restricting the connectivity pattern between queries and keys, sparse attention achieves linear or near-linear complexity— $O(n \cdot w)$ for sliding windows, $O(n\sqrt{n})$ for the strided and fixed patterns of the Sparse Transformer, and $O(n \log n)$ for hashing-based schemes such as Reformer—while preserving much of the representational capacity of full attention.

The mathematical framework presented in this paper unifies several influential sparse attention architectures. The key insight is that the softmax normalization need only be computed over a carefully chosen subset S_i of positions for each query i . The choice of S_i encodes domain-specific inductive biases: sliding windows exploit temporal locality, global attention provides aggregation points for summary information, dilated patterns extend receptive fields without increasing computational cost, and block-sparse patterns match the way hardware performs matrix multiplication.

Domain-Specific Applications. The effectiveness of sparse attention patterns depends critically on the structure of the underlying data. For natural language, local context within a sentence combined with document-level global tokens (as in Longformer) captures both fine-grained syntax and broad semantic dependencies. For genomic sequences, strided patterns can capture regulatory elements separated by thousands of base pairs. For financial time series, sliding windows align naturally with the temporal locality of market microstructure.

Financial Time Series and HFT. High-frequency trading applications present an ideal use case for sparse attention. Order book dynamics exhibit strong temporal locality: the current state depends primarily on recent ticks, with occasional dependency on regime context further back in history. The causal local-plus-dilated pattern described above captures both, and it can process sequences of 64K ticks (3–4 hours of trading) with modest memory: with the default parameters and four heads at 32-bit precision, the attention scores occupy about 0.6 GB, compared to roughly 64 GB (16 GB per head) for a dense attention matrix. Its compute falls by the same factor of roughly 110. This brings sub-millisecond inference within reach for full-session context, although end-to-end latency depends on the kernel implementation and remains to be confirmed by benchmarks.

Implementation Considerations. While this paper focuses on mathematical foundations, practical implementations require careful attention to computational efficiency. Sparse attention patterns enable significant asymptotic improvements, but realizing these gains requires kernel-level optimizations: fused attention operations to minimize memory traffic, blocked computation to maximize cache reuse, and specialized data structures to avoid irregular memory access patterns. Recent systems such as NSA and MoBA show that when the sparsity pattern is designed around these constraints from the start, sparse attention delivers real wall-clock speedups at long context lengths, not only lower operation counts. These implementation details, while beyond our scope here, are critical for achieving production-grade performance.

The continued development of sparse attention mechanisms—and their integration with complementary techniques like linear attention, low-rank approximations, and IO-aware algorithms—promises to extend Transformer applicability to increasingly long sequences and latency-sensitive domains.

References

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. *Attention is all you need*. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [2] R. Child, S. Gray, A. Radford, and I. Sutskever. *Generating long sequences with sparse transformers*. arXiv preprint arXiv:1904.10509, 2019.
- [3] I. Beltagy, M. E. Peters, and A. Cohan. *Longformer: The long-document transformer*. arXiv preprint arXiv:2004.05150, 2020.
- [4] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed. *Big bird: Transformers for longer sequences*. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.
- [5] N. Kitaev, L. Kaiser, and A. Levskaya. *Reformer: The efficient transformer*. In *International Conference on Learning Representations (ICLR)*, 2020.
- [6] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma. *Linformer: Self-attention with linear complexity*. arXiv preprint arXiv:2006.04768, 2020.
- [7] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré. *FlashAttention: Fast and memory-efficient exact attention with IO-awareness*. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

- [8] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis. *Efficient streaming language models with attention sinks*. In *International Conference on Learning Representations (ICLR)*, 2024.
- [9] J. Yuan, H. Gao, D. Dai, J. Luo, L. Zhao, Z. Zhang, Z. Xie, et al. *Native sparse attention: Hardware-aligned and natively trainable sparse attention*. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2025.
- [10] E. Lu, Z. Jiang, J. Liu, Y. Du, T. Jiang, C. Hong, S. Liu, W. He, et al. *MoBA: Mixture of block attention for long-context LLMs*. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.