

## Seminar Report

---

# Dynamic and Adaptive Attention in Modern Transformers

---

Naveen Reddy Parimireddy

MatrNr: 21676609

Supervisor: Lauritz Rasbach

Georg-August-Universität Göttingen  
Institute of Computer Science

September 2026

# Declaration on the Use of AI and Third-Party Assistance

## Overview

- Writing
  - L<sup>A</sup>T<sub>E</sub>X, compiled via Overleaf.
  - An AI language model (Anthropic Claude) was used to draft the written content of this report, including the prose explanations, technical derivations, tables, figures, and citation integration. This drafting was based on the author’s understanding of the topic, developed through the accompanying seminar presentation and directed study of the underlying literature. The author selected the topic and source papers, directed the structure and scope of the report across multiple review passes, specified corrections including citation formatting and consistency between sections, and reviewed and approved all technical content, citations, and conclusions prior to submission.
- Images
  - Not applicable; no externally sourced images are included in this report.
- Diagrams
  - All diagrams in this report were produced directly in TikZ within the L<sup>A</sup>T<sub>E</sub>X source. The diagrams were drafted with AI assistance based on figures the author had already prepared for the accompanying seminar presentation, and were adapted to the report’s schematic style.
- Code
  - Not applicable; see Appendix B.
- AI
  - Anthropic Claude was used to draft the prose, tables, figures, and citation structure of this report, and to assist with L<sup>A</sup>T<sub>E</sub>X, TikZ, and cross-references. The author supplied the underlying technical understanding and source literature, directed the report’s structure and content across several rounds of revision, identified and requested corrections to specific inconsistencies (for example, aligning Table 3 with the claims made in Section 2.5), and reviewed and verified all AI-generated content for technical accuracy before submission.
- Third-Party
  - The report was prepared under the formative supervision of Lauritz Rasbach, in accordance with the standard seminar supervision process described in the course guidelines. No other third-party contributions were made.

I hereby declare that all uses of tools and AI as well as assistance from third parties in this work have been fully and accurately disclosed above.

Missing or incorrect information may be considered an attempt to cheat.

# Abstract

The Transformer architecture and its self-attention mechanism have become a central paradigm for modern sequence modelling because they allow tokens to interact directly across an input sequence and thereby capture long-range dependencies. This flexibility, however, introduces quadratic computational cost in sequence length,  $\mathcal{O}(n^2)$ , while explicitly materialising the attention matrix also incurs quadratic memory cost. These costs make long-context processing increasingly challenging as application context windows extend to hundreds of thousands or millions of tokens. In addition, autoregressive inference introduces a separate memory constraint through the key-value (KV) cache, whose size grows with the number of cached tokens. These limitations are particularly relevant to high-performance data analytics workloads involving long documents, code repositories, and other large-context inputs.

This report investigates how modern attention and sequence-modelling strategies address these computational and memory constraints, and how their trade-offs in complexity, memory usage, exactness, and long-range information access affect their suitability for different deployment scenarios. The surveyed approaches are organised into three aspects: approximation-based attention, represented by Linformer and Reformer; structural sparsity, represented by Longformer; and hardware- and architecture-aware approaches, including FlashAttention, hybrid Transformer–State Space Model architectures such as Jamba, and inference-time KV-cache optimisation methods such as H2O and SparQ.

The comparison shows that these approaches address different parts of the long-context bottleneck rather than providing a single general solution. Approximation and sparse-attention methods reduce the amount of computation or the number of token interactions, but introduce architectural or information-access trade-offs. FlashAttention preserves the exact attention computation while substantially reducing memory traffic, but does not change the quadratic asymptotic complexity of attention. Hybrid architectures and KV-cache optimisation methods address complementary inference-time constraints by changing the sequence-processing architecture or reducing the amount of cached information associated with dense attention. Overall, the literature indicates that effective long-context deployment increasingly depends on matching the computational strategy to the workload, memory constraints, and required level of direct long-range information access.

# Contents

|                                                                |           |
|----------------------------------------------------------------|-----------|
| List of Tables                                                 | iv        |
| List of Figures                                                | iv        |
| List of Abbreviations                                          | v         |
| <b>1 Introduction</b>                                          | <b>1</b>  |
| <b>2 Background and Terminology</b>                            | <b>2</b>  |
| 2.1 From Recurrent Networks to Attention . . . . .             | 2         |
| 2.2 Scaled Dot-Product Attention . . . . .                     | 2         |
| 2.3 The Quadratic Bottleneck . . . . .                         | 3         |
| 2.4 The Key-Value Cache Memory Wall . . . . .                  | 4         |
| 2.5 Two Phases of Inference: Prefilling and Decoding . . . . . | 4         |
| <b>3 Aspect 1: Approximation-Based Attention</b>               | <b>5</b>  |
| 3.1 Low-Rank Projections: Linformer . . . . .                  | 5         |
| 3.2 Content-Based Hashing: Reformer . . . . .                  | 6         |
| <b>4 Aspect 2: Structural Sparsity</b>                         | <b>7</b>  |
| <b>5 Aspect 3: Hardware- and Architecture-Aware Approaches</b> | <b>8</b>  |
| 5.1 FlashAttention . . . . .                                   | 8         |
| 5.2 Hybrid State Space Models: Jamba . . . . .                 | 9         |
| 5.3 Inference-Time Key-Value Cache Optimisation . . . . .      | 10        |
| <b>6 Discussion</b>                                            | <b>12</b> |
| 6.1 Comparative Trade-offs . . . . .                           | 12        |
| 6.2 Empirical Time and Accuracy Comparison . . . . .           | 14        |
| 6.3 Characteristic Limitations . . . . .                       | 14        |
| <b>7 Conclusion</b>                                            | <b>15</b> |
| <b>References</b>                                              | <b>16</b> |
| <b>A Work Sharing</b>                                          | <b>C1</b> |
| <b>B Code Samples</b>                                          | <b>C1</b> |
| <b>C Transparency Report</b>                                   | <b>C1</b> |

# List of Tables

|   |                                                                                                                                                                                                                                   |    |
|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 1 | Illustrative growth of the pairwise attention computation with sequence length. . . . .                                                                                                                                           | 3  |
| 2 | Illustrative key-value cache memory requirements computed from Equation (3) for two representative model configurations ( $h_{KV} = 8$ , $d = 128$ , 16-bit Floating Point (FP16)). . . . .                                       | 4  |
| 3 | Representative empirical efficiency and quality results reported by the surveyed papers. The values are not directly comparable across methods because the evaluation benchmarks, hardware, models, and workloads differ. . . . . | 12 |
| 4 | Comparison of the long-context strategies covered in this report, including their primary target inference phase. . . . .                                                                                                         | 13 |

# List of Figures

|   |                                                    |    |
|---|----------------------------------------------------|----|
| 1 | Scaled dot-product attention . . . . .             | 3  |
| 2 | Linformer inference speedup . . . . .              | 6  |
| 3 | Reformer speed scaling . . . . .                   | 7  |
| 4 | Longformer attention pattern . . . . .             | 7  |
| 5 | Longformer character-modelling results . . . . .   | 8  |
| 6 | FlashAttention speedup across benchmarks . . . . . | 9  |
| 7 | Jamba throughput at long context . . . . .         | 10 |
| 8 | H2O throughput improvement . . . . .               | 11 |

# List of Abbreviations

|             |                                                         |
|-------------|---------------------------------------------------------|
| <b>BERT</b> | Bidirectional Encoder Representations from Transformers |
| <b>FP16</b> | 16-bit Floating Point                                   |
| <b>GPU</b>  | Graphics Processing Unit                                |
| <b>GQA</b>  | Grouped Query Attention                                 |
| <b>H2O</b>  | Heavy-Hitter Oracle                                     |
| <b>HBM</b>  | High-Bandwidth Memory                                   |
| <b>LSH</b>  | Locality-Sensitive Hashing                              |
| <b>LSTM</b> | Long Short-Term Memory                                  |
| <b>SRAM</b> | Static Random-Access Memory                             |
| <b>SSM</b>  | State Space Model                                       |

# 1 Introduction

The ability to process long input sequences has become increasingly important for high-performance data analytics. Applications include reasoning over code repositories, analysing large legal or financial documents, and processing extensive collections of textual data. The self-attention mechanism introduced by Vaswani et al. [Vas+17] enabled this class of applications by replacing the sequential hidden-state updates of recurrent architectures with direct interactions between tokens. This design allows tokens to access information from distant positions directly, without requiring it to pass through a long sequence of recurrent state updates, and forms the foundation of the Transformer architecture used in modern sequence modelling.

The expressive power of self-attention, however, comes with substantial computational and memory costs. These costs manifest differently across the two principal stages of Transformer inference: *prefilling*, in which an input prompt is processed as a whole, and *decoding*, in which output tokens are generated one at a time. As we discuss in Section 2.5, prefilling is typically limited by the amount of computation available, whereas decoding is typically limited by the memory bandwidth needed to access previously computed representations. Because every token can interact with every other token, the attention computation has quadratic complexity in the sequence length  $n$ , denoted by  $\mathcal{O}(n^2)$ . Explicitly materialising the  $n \times n$  attention matrix also incurs quadratic memory cost. These costs were manageable for the relatively short context lengths used by early Transformer models such as Bidirectional Encoder Representations from Transformers (BERT), which supported sequences of up to 512 tokens [Dev+19]. As context windows increase to hundreds of thousands or millions of tokens, however, the quadratic scaling becomes an increasingly important limitation. In addition to the computation required during prompt processing, autoregressive generation introduces a separate memory constraint through the growing key-value cache. At sufficiently large context lengths, these computational and memory requirements can exceed the capacity of a single accelerator.

## Research Question

This report investigates how modern attention and sequence-modelling strategies reduce the computational and memory costs associated with long-context processing, and how their respective trade-offs in algorithmic complexity, memory usage, exactness, and long-range information access affect their suitability for different deployment scenarios.

## Contributions

This report makes the following contributions.

- We provide a self-contained derivation of the computational and memory bottlenecks of standard self-attention, distinguishing the prefilling and decoding phases and illustrating the growth of the key-value cache with a concrete case study.
- We organise representative efficiency strategies from the literature into three aspects: approximation-based attention, structural sparsity, and hardware- and architecture-aware approaches.

- We compare these strategies according to their computational scaling, exactness, memory requirements, empirical efficiency, and practical limitations, and discuss the conditions under which different approaches are appropriate for long-context deployment.

## Report Structure

Section 2 establishes the mathematical foundation of scaled dot-product attention and explains the computational and memory bottlenecks that motivate the remainder of the report. Section 3 presents approximation-based strategies, namely Linformer and Reformer. Section 4 presents Longformer’s structural sparsity approach. Section 5 presents hardware- and architecture-aware approaches, covering FlashAttention, hybrid State Space Model architectures, and inference-time key-value cache optimisation. Section 6 compares the approaches and discusses their respective trade-offs and limitations. Finally, Section 7 summarises the main findings and outlines open challenges.

# 2 Background and Terminology

## 2.1 From Recurrent Networks to Attention

Prior to the Transformer, sequence models predominantly relied on recurrent architectures, including Long Short-Term Memory (LSTM) networks [HS97]. These models process a sequence one token at a time, maintaining a fixed-size hidden state that is updated at every step. This design has two structural limitations. First, gradient signals can decay across many time steps, which limits the effective range over which a recurrent model can learn dependencies. Second, because each hidden-state update depends on the previous one, the computation cannot be fully parallelised across the sequence, which limits the utilisation of modern parallel hardware. Self-attention addresses both limitations by allowing tokens to interact directly while enabling the processing of the sequence in parallel.

## 2.2 Scaled Dot-Product Attention

Given an input sequence of  $n$  tokens represented by an embedding matrix  $X \in \mathbb{R}^{n \times d}$ , self-attention computes three learned linear projections of the input:

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V, \quad Q, K \in \mathbb{R}^{n \times d_k}, \quad V \in \mathbb{R}^{n \times d_v} \quad (1)$$

The *Query* matrix  $Q$  encodes what each token seeks; the *Key* matrix  $K$  encodes what each token offers as a match target; and the *Value* matrix  $V$  encodes the content that each token contributes to the output. Attention is then computed as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V. \quad (2)$$

Figure 1 illustrates the data flow of Equation (2). The dot product  $QK^\top$  produces an  $n \times n$  matrix in which entry  $(i, j)$  measures the relevance of token  $j$  to token  $i$ . Dividing by  $\sqrt{d_k}$  prevents the dot-product scores from growing excessively with the dimensionality of the representations, which helps avoid excessively peaked softmax distributions [Vas+17].

The softmax normalises each row into a probability distribution, and the final matrix multiplication with  $V$  produces, for every token, a weighted combination of the value vectors of the sequence.

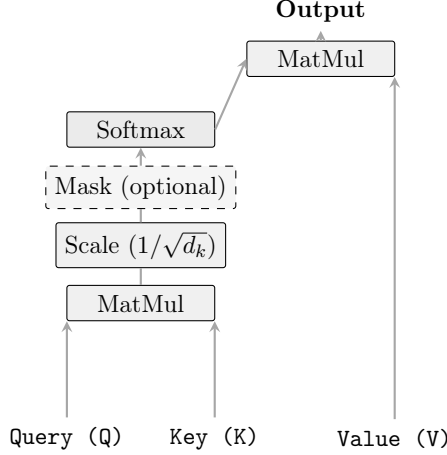


Figure 1: Data flow of scaled dot-product attention as defined in Equation (2). The dashed mask step is applied only in causal (autoregressive) settings. Diagram redrawn after the original architecture description in Vaswani et al. [Vas+17].

In practice, the Transformer applies  $h$  attention computations in parallel, termed *heads*, each operating on a projection of dimension  $d_k = d/h$  [Vas+17]. Multi-head attention allows different heads to learn different relationships between tokens while maintaining parallel computation across the sequence.

## 2.3 The Quadratic Bottleneck

The dominant pairwise computation in Equation (2) is the matrix product  $QK^\top$ , which requires  $\mathcal{O}(n^2 d_k)$  floating-point operations. Because this cost grows with the square of the sequence length, doubling the input length approximately quadruples the number of pairwise interactions. Table 1 illustrates the practical consequence of this scaling across representative context lengths.

Table 1: Illustrative growth of the pairwise attention computation with sequence length.

| Sequence-length scenario     | Tokens $n$ | Pairwise terms ( $n^2$ ) |
|------------------------------|------------|--------------------------|
| BERT context length [Dev+19] | 512        | 262,144                  |
| 128K-token context           | 128,000    | 16.4 billion             |
| 2M-token context             | 2,000,000  | 4 trillion               |

Beyond the arithmetic cost, the  $n \times n$  attention matrix can require substantial memory when materialised explicitly. For example, at  $n = 128,000$ , storing the matrix in 32-bit precision requires approximately  $128,000^2 \times 4$  bytes, or about 65 GB, for a single layer. Such memory requirements can become prohibitive depending on the accelerator and implementation, motivating techniques that avoid materialising the full attention matrix.

## 2.4 The Key-Value Cache Memory Wall

During autoregressive generation, a model produces one output token at a time. To avoid recomputing the Key and Value projections for every previously generated token at every step, these projections are stored in a *key-value cache*. The memory footprint of this cache, in FP16 precision, is given by

$$\text{Memory per token} = 2 \times 2 \times L \times h_{KV} \times d, \quad (3)$$

where the first factor of 2 accounts for the two bytes per FP16 value, the second factor of 2 accounts for storing both the Key and Value,  $L$  is the number of Transformer layers,  $h_{KV}$  is the number of key-value heads retained under Grouped Query Attention (GQA), and  $d$  is the dimensionality of each head.

Table 2 applies Equation (3) to two representative configurations in the Llama 3 family, illustrating how cache requirements scale with model depth and context length. The Llama 3 report specifies 32 layers for the 8B model, 80 layers for the 70B model, and 8 key-value heads for both configurations [Lla24].

Table 2: Illustrative key-value cache memory requirements computed from Equation (3) for two representative model configurations ( $h_{KV} = 8$ ,  $d = 128$ , FP16).

| Configuration                    | Layers $L$ | Per-token cache | At 1M-token context |
|----------------------------------|------------|-----------------|---------------------|
| Smaller configuration (8B-scale) | 32         | 128 KiB         | 128 GiB             |
| Larger configuration (70B-scale) | 80         | 320 KiB         | 320 GiB             |

At a context length of one million tokens, the larger configuration requires approximately 320 GiB of dedicated memory for the key-value cache alone. This exceeds the memory capacity of many individual accelerators and illustrates why KV-cache management becomes an important constraint for long-context inference.

## 2.5 Two Phases of Inference: Prefilling and Decoding

Transformer inference proceeds in two distinct phases, each characterised by a different limiting factor. During *prefilling*, the complete input sequence is processed in parallel, and the dominant cost is the computation associated with the pairwise interactions between all input tokens; prefilling is consequently limited primarily by the available compute throughput of the accelerator. During *decoding*, tokens are generated sequentially, and previously computed Key and Value representations are retrieved from the key-value cache at every step; as the context grows, the volume of data that must be transferred from memory can come to dominate execution time, so that decoding is limited primarily by memory bandwidth rather than raw compute.

The surveyed methods address these two phases in different ways. Linformer, Reformer, and Longformer reduce the number of token interactions required by modifying the attention computation or its connectivity pattern, primarily reducing the compute cost of prefilling. FlashAttention preserves the exact mathematical attention operation while reducing the movement of intermediate data between different levels of the GPU memory hierarchy, benefiting both phases but particularly prefilling. H2O and SparQ instead target the inference-time KV cache directly, reducing the amount of cached information that must be stored or transferred during decoding. Hybrid architectures such as Jamba address the problem at the architectural level by replacing many attention layers with a

linear-time State Space mechanism while retaining a smaller number of attention layers, easing the memory-bandwidth burden of decoding in particular.

## 3 Aspect 1: Approximation-Based Attention

Approximation-based methods retain the general structure of Equation (2) but replace the exhaustive  $n \times n$  computation with a cheaper approximation, accepting approximation error in exchange for reduced computational or memory requirements.

### 3.1 Low-Rank Projections: Linformer

Wang et al. [Wan+20] analysed attention matrices and argued that their effective rank can be substantially lower than the sequence length. This observation motivates approximating the full  $n \times n$  attention computation using a lower-dimensional representation.

Linformer introduces two projection matrices  $E, F \in \mathbb{R}^{K \times n}$ , where  $K \ll n$  is a fixed projection dimension, and projects the Key and Value matrices before the attention computation:

$$\text{Attention}_{\text{Linear}}(Q, K, V) = \text{softmax}\left(\frac{Q(EK)^\top}{\sqrt{d_k}}\right)(FV). \quad (4)$$

Because  $E$  and  $F$  have a fixed number of rows independent of  $n$ , the dominant pairwise computation becomes  $\mathcal{O}(nKd_k)$  rather than  $\mathcal{O}(n^2d_k)$ . For fixed  $K$  and  $d_k$ , this gives linear scaling with sequence length. For example, if  $n = 128,000$  and  $K = 256$ , the number of positions participating in the projected interaction is substantially smaller than the original sequence length.

This efficiency comes with architectural constraints. The projection matrices are defined with respect to the sequence dimension, which means that a model configured for a particular sequence length cannot simply be resized to an arbitrary new length without modifying the projection mechanism or retraining the model. The low-rank approximation also assumes that the relevant attention structure can be represented accurately in the lower-dimensional projected space. Consequently, tasks whose performance depends on fine-grained or highly selective token interactions may expose limitations of the approximation. The suitability of Linformer therefore depends on whether the low-rank structure assumed by the method is appropriate for the target workload.

**Empirical results.** Wang et al. [Wan+20] report inference speedups of  $1.5\times$  at a sequence length of 512 tokens, rising to  $20\times$  at 65,536 tokens, together with memory reductions of  $1.7\times$  to  $60\times$  over the same range, all measured at a fixed projection dimension  $k = 128$  (Figure 2). On the GLUE benchmark, the projected model achieves an average score of 92.30, compared to 92.25 for RoBERTa-base, indicating near-parity on this benchmark.

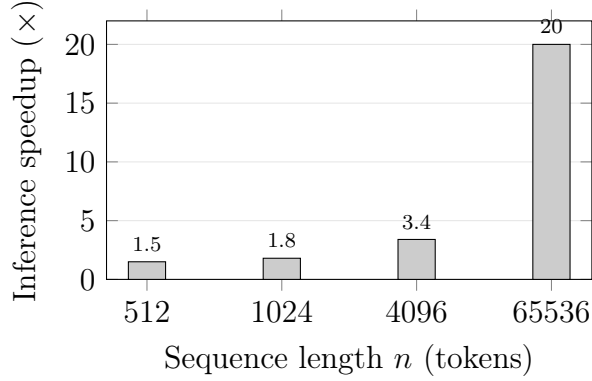


Figure 2: Inference speedup of Linformer over standard attention at a fixed projection dimension  $k = 128$ , with values redrawn from results reported by Wang et al. [Wan+20].

### 3.2 Content-Based Hashing: Reformer

Kitaev, Kaiser, and Levskaya [KKL20] proposed replacing exhaustive pairwise comparison with a locality-sensitive hashing mechanism. The underlying observation is that tokens that have similar Query and Key representations are more likely to require direct interaction. Locality-Sensitive Hashing (LSH) can therefore be used to identify candidate token pairs without comparing every token with every other token.

Reformer projects token representations onto a set of random hyperplanes and uses the resulting hash codes to assign tokens to buckets. Attention is then computed within the resulting buckets rather than across the complete sequence. Because a single hashing round can separate two relevant tokens, multiple hashing rounds can be used to increase the probability that relevant token pairs are placed in a common bucket.

Sorting tokens according to their hash buckets and computing attention within the resulting groups gives a complexity of approximately  $\mathcal{O}(n \log n)$ , improving on the quadratic cost of standard attention. The main limitation is that the attention pattern depends on the quality of the hash-based candidate selection. Even with multiple hashing rounds, tokens that should interact may fail to appear in a common bucket, introducing approximation error. Increasing the number of hashing rounds can improve the probability of recovering relevant interactions, but also increases computational cost. Reformer therefore trades exhaustive token comparison for a probabilistic mechanism whose effectiveness depends on the structure of the attention patterns encountered by the model.

**Empirical results.** Kitaev, Kaiser, and Levskaya [KKL20] report that, on a synthetic duplication task, an LSH attention model using four hashing rounds reaches 99.9% accuracy, closely approaching the 100% achieved by full attention on the same task, while inference speed remains approximately flat as sequence length increases, in contrast to standard attention, which slows sharply at longer sequences (Figure 3). On a machine translation task, a reversible Reformer model reaches a BLEU score of 27.6, compared to 27.3 for the baseline Transformer of Vaswani et al. [Vas+17], showing similar translation quality in the reported setting.

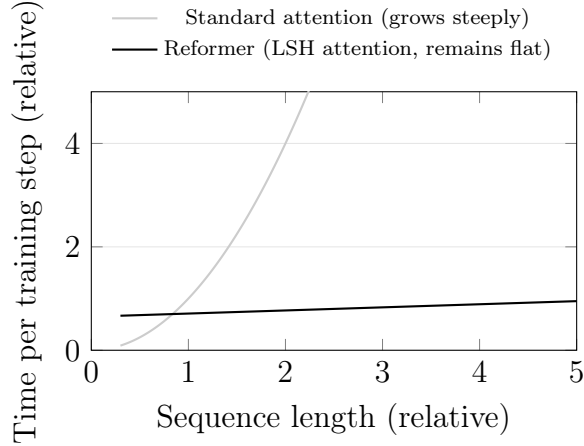


Figure 3: Qualitative scaling behaviour of per-step training time with sequence length. Standard attention grows sharply, while LSH attention remains approximately flat. The curves are illustrative rather than exact reproductions of the measurements reported by Kitaev, Kaiser, and Levskaya [KKL20].

## 4 Aspect 2: Structural Sparsity

Rather than approximating the attention matrix numerically, structural sparsity methods restrict, by construction, which token pairs are permitted to interact. The restriction is based on prior assumptions about which interactions are likely to be useful.

Beltagy, Peters, and Cohan [BPC20] proposed Longformer, which combines local attention with selected global attention. The model uses two complementary patterns: local interactions between nearby tokens and global interactions involving a small set of designated positions that require access to the full sequence. This structure is illustrated schematically in Figure 4.



Figure 4: Schematic attention mask used by Longformer, combining a fixed-width sliding window (light grey diagonal band) with global tokens that attend to, and are attended by, all positions in the sequence (dark grey row and column). Diagram redrawn after Beltagy, Peters, and Cohan [BPC20].

Under sliding window attention, each token attends only to a fixed number of nearby positions. For a window size  $w$ , the resulting attention computation has a total complexity of  $\mathcal{O}(nw)$ , which is linear in  $n$  for fixed  $w$ . Under global attention, a small number  $g$  of designated tokens attend to, and are attended to, all positions in the sequence, adding a further  $\mathcal{O}(ng)$  cost. The combined complexity remains linear in sequence length when  $w$  and  $g$  are fixed.

The principal limitation of this design is that it restricts direct token-to-token interactions to the chosen attention pattern: a token can access information outside its local window only indirectly, through subsequent layers or through designated global tokens.

Consequently, the effectiveness of Longformer depends on the choice of window size, attention pattern, and placement of global tokens. If an application requires precise interactions between distant positions that are not directly connected by the selected pattern, information may need to propagate through multiple layers before it can be combined. The fixed attention structure therefore introduces a trade-off between computational efficiency and the flexibility of unrestricted global attention.

**Empirical results.** Beltagy, Peters, and Cohan [BPC20] report a bits-per-character score of 1.00 on the enwik8 character-modelling benchmark, compared with 1.11 reported for the T12 model, and a similar improvement from 1.18 to 1.10 bits-per-character on the text8 benchmark (Figure 5). On the WikiHop question-answering benchmark, the authors report an accuracy of 73.8–75.0%, compared to 72.4% for RoBERTa-base restricted to a 512-token context. These results illustrate the task-dependent effect of the longer contexts enabled by Longformer.

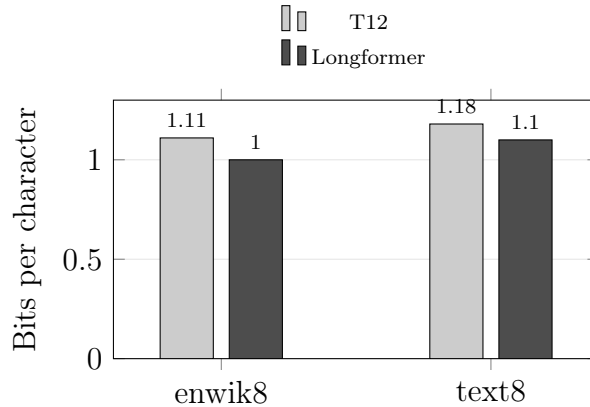


Figure 5: Bits-per-character on the enwik8 and text8 benchmarks for T12 versus Longformer (lower is better), with values redrawn from results reported by Beltagy, Peters, and Cohan [BPC20].

## 5 Aspect 3: Hardware- and Architecture-Aware Approaches

A conceptually distinct family of strategies either preserves the exact mathematical formulation of attention while optimising its execution, or changes the sequence-processing architecture itself to reduce the computational and memory requirements associated with dense attention.

### 5.1 FlashAttention

Modern Graphics Processing Units (GPUs) contain multiple memory levels with different capacities and bandwidths. High-Bandwidth Memory (HBM) provides large capacity but is slower to access than the small, fast on-chip Static Random-Access Memory (SRAM). A conventional implementation of attention repeatedly transfers intermediate attention

data between these memory levels. At large sequence lengths, this data movement can become a major component of execution time and memory consumption.

Dao et al. [Dao+22] addressed this problem through an IO-aware implementation of exact attention. FlashAttention partitions  $Q$ ,  $K$ , and  $V$  into tiles that fit into fast on-chip memory, computes attention for each tile pair locally, and accumulates the output using numerically stable running softmax statistics. The full  $n \times n$  attention matrix therefore does not need to be materialised in HBM.

Crucially, this reformulation does not approximate the attention function. FlashAttention computes the same mathematical attention operation as standard attention, up to floating-point rounding. It therefore improves memory efficiency and practical execution time without changing the underlying  $\mathcal{O}(n^2)$  asymptotic complexity. The reported performance improvements arise primarily from reducing memory traffic and improving accelerator utilisation rather than from reducing the number of pairwise attention operations.

**Empirical results.** Dao et al. [Dao+22] report end-to-end wall-clock speedups of 15% on BERT-large training,  $3\times$  on GPT-2 training, and  $2.4\times$  on the Long-Range Arena benchmark suite (Figure 6), while computing exact attention rather than an approximation. The longer context lengths that FlashAttention makes tractable also yield downstream quality improvements in the evaluated settings: the authors report a 0.7-point improvement in GPT-2 perplexity and a 6.4-point improvement on a long-document classification task.

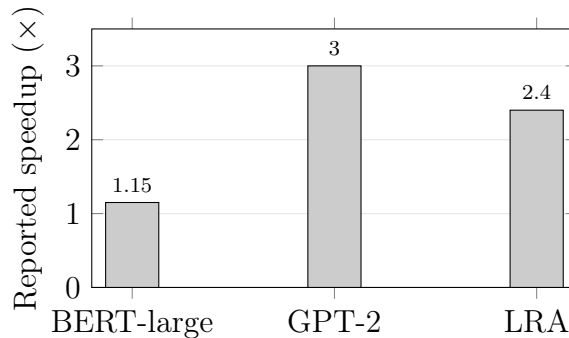


Figure 6: Reported end-to-end wall-clock speedups for FlashAttention in the BERT-large, GPT-2, and Long-Range Arena evaluations, with values redrawn from Dao et al. [Dao+22].

## 5.2 Hybrid State Space Models: Jamba

An alternative to optimising the execution of attention is to replace a substantial portion of the attention layers with an architecturally different sequence-processing mechanism. State Space Models (SSMs), exemplified by Mamba [GD23], process sequences through a recurrent state representation. A simplified formulation can be written as

$$h_t = Ah_{t-1} + Bx_t, \quad y_t = Ch_t, \quad (5)$$

where  $h_t$  represents the internal state at position  $t$ . SSM-based sequence processing can scale linearly with sequence length and does not require an attention-style KV cache that grows with the number of tokens. Its fixed-size state representation, however, differs fundamentally from the unrestricted pairwise interactions provided by attention.

Information from earlier positions must be incorporated into the evolving state rather than being directly retrieved through a full attention operation.

Lieber et al. [Lie+24] proposed Jamba, a hybrid architecture that combines Mamba layers with standard Transformer attention layers. The original Jamba configuration interleaves one attention layer with seven Mamba layers. In its 32-layer backbone, this results in four attention layers and twenty-eight Mamba layers [Lie+24]. The architecture therefore retains a small number of attention layers for operations that benefit from direct token-to-token interactions while using Mamba layers for the majority of sequence processing.

This hybrid design changes the computational profile of the model rather than simply replacing quadratic attention with a linear alternative. The Mamba components scale linearly with sequence length, whereas the retained attention components retain the quadratic attention computation and associated KV-cache requirements. Because only a subset of layers uses attention, however, the amount of attention-associated cache can be substantially smaller than in a dense Transformer with the same depth.

The main trade-off is that SSM and attention layers provide different forms of information access. SSM layers maintain a compressed recurrent state, while attention layers permit direct access to previous token representations. The hybrid architecture therefore does not make attention unnecessary; instead, it allocates attention selectively where direct token interactions are particularly useful.

**Empirical results.** Lieber et al. [Lie+24] report a  $3\times$  higher throughput than Mixtral 8x7B when serving a 128K-token context on 4 A100 GPUs, and demonstrate stable operation at context lengths of up to 256,000 tokens, a regime in which a comparable dense model such as Llama-2-70B does not fit in the same hardware configuration (Figure 7). On standard academic benchmarks, Jamba shows mixed results across tasks, with higher scores on some benchmarks and lower scores on others relative to the cited baselines.

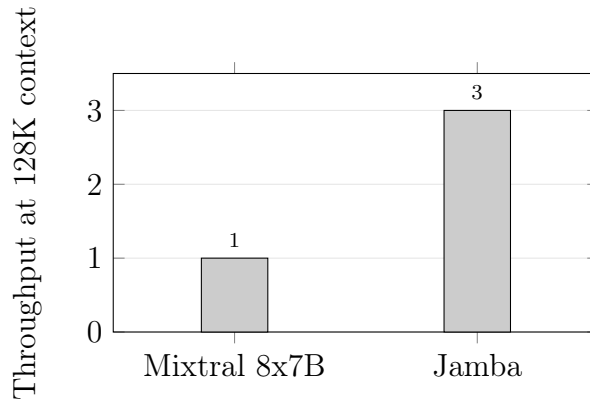


Figure 7: Relative inference throughput of Jamba versus Mixtral 8x7B at a 128K-token context, with values redrawn from results reported by Lieber et al. [Lie+24].

### 5.3 Inference-Time Key-Value Cache Optimisation

A further class of methods operates purely at inference time, without modifying the model architecture, by managing the key-value cache described in Section 2.4 more efficiently. Zhang et al. [Zha+23] observe that a small subset of tokens can contribute

disproportionately to attention and propose Heavy-Hitter Oracle (H2O), which dynamically retains recent tokens and heavy-hitter tokens while evicting less important entries. This bounds cache growth while attempting to preserve the information most relevant to subsequent predictions.

Ribar et al. [Rib+24] instead target the bandwidth required to retrieve cached representations. SparQ uses a lightweight query-based selection mechanism to identify relevant cache entries before transferring them from memory, thereby reducing the amount of cache data that needs to be accessed. These methods illustrate that KV-cache management can be treated as a separate inference-time optimisation problem from the asymptotic complexity of the underlying attention mechanism.

**Empirical results.** Zhang et al. [Zha+23] report throughput improvements of up to  $29\times$  over DeepSpeed Zero-Inference and HuggingFace Accelerate, and up to  $3\times$  over FlexGen, together with up to  $1.9\times$  lower latency at a fixed batch size, while retaining only 20% of the KV cache with no measurable accuracy degradation (Figure 8). Ribar et al. [Rib+24] report up to  $8\times$  savings in attention data transfer across several evaluated language models, with negligible reported accuracy loss, achieved without any retraining of the underlying model.

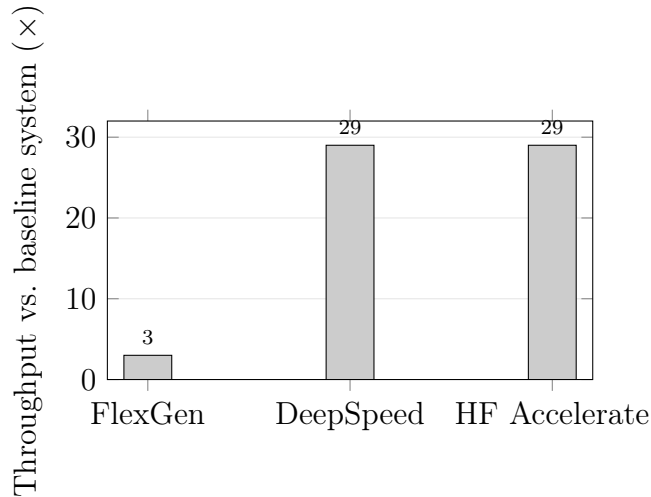


Figure 8: Inference throughput improvement of H2O relative to three baseline inference systems, with values redrawn from results reported by Zhang et al. [Zha+23].

Table 3: Representative empirical efficiency and quality results reported by the surveyed papers. The values are not directly comparable across methods because the evaluation benchmarks, hardware, models, and workloads differ.

| Method         | Evaluation setting                               | Reported efficiency result                                                                                                                   | Reported quality result                                                                                   |
|----------------|--------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------|
| Linformer      | 512–65,536 tokens; GLUE                          | $1.5\times$ – $20\times$ inference speedup and $1.7\times$ – $60\times$ memory reduction                                                     | GLUE average score 92.30 vs. 92.25 for RoBERTa-base [Wan+20]                                              |
| Reformer       | Synthetic duplication; machine translation       | Approximately flat inference-time scaling with sequence length in the reported duplication experiment                                        | 99.9% vs. 100% accuracy on duplication; BLEU 27.6 vs. 27.3 [KKL20]                                        |
| Longformer     | enwik8, text8, WikiHop                           | Structured attention with $\mathcal{O}(n)$ scaling for fixed local and global attention parameters                                           | 1.00 vs. 1.11 BPC on enwik8 (T12); 1.10 vs. 1.18 BPC on text8 (T12); WikiHop 73.8–75.0% vs. 72.4% [BPC20] |
| FlashAttention | BERT-large, GPT-2, LRA                           | $1.15\times$ , $3\times$ , and $2.4\times$ wall-clock speedup, respectively                                                                  | Exact attention; reported downstream quality gains in longer-context settings [Dao+22]                    |
| Jamba          | 128K context on 4 A100 GPUs; academic benchmarks | $3\times$ throughput versus Mixtral 8x7B at 128K context                                                                                     | Mixed results across benchmarks, with higher scores on some tasks and lower scores on others [Lie+24]     |
| H2O            | Inference under KV-cache pressure                | Up to $29\times$ throughput versus DeepSpeed Zero-Inference and HuggingFace Accelerate, $3\times$ versus FlexGen, and 20% KV-cache retention | No measurable accuracy degradation in the reported setting [Zha+23]                                       |
| SparQ          | Several evaluated language models                | Up to $8\times$ savings in attention data transfer                                                                                           | Negligible reported accuracy loss [Rib+24]                                                                |

## 6 Discussion

### 6.1 Comparative Trade-offs

Table 4 summarises the strategies presented in Sections 3 to 5 along five dimensions: computational scaling, computation type, target inference phase, principal advantage, and principal limitation.

Table 4: Comparison of the long-context strategies covered in this report, including their primary target inference phase.

| Method             | Complexity                                          | Computation type                  | Target phase | Principal advantage                                                      | Principal limitation                                          |
|--------------------|-----------------------------------------------------|-----------------------------------|--------------|--------------------------------------------------------------------------|---------------------------------------------------------------|
| Standard attention | $\mathcal{O}(n^2)$                                  | Exact attention                   | Both         | Unrestricted global token-to-token interactions                          | Quadratic scaling with sequence length                        |
| Linformer          | $\mathcal{O}(n)$                                    | Approximate attention             | Prefilling   | Low-rank linear scaling                                                  | Depends on the low-rank assumption and sequence configuration |
| Reformer           | $\mathcal{O}(n \log n)$                             | Approximate sparse attention      | Prefilling   | Reduces exhaustive token comparisons                                     | Hash-based candidate selection can miss relevant interactions |
| Longformer         | $\mathcal{O}(n)$                                    | Structured sparse attention       | Prefilling   | Linear scaling with local and global attention patterns                  | Restricted attention connectivity                             |
| Flash Attention    | $\mathcal{O}(n^2)$                                  | Exact attention                   | Both         | Reduces memory traffic and improves hardware efficiency                  | Quadratic asymptotic complexity remains                       |
| Jamba              | $\mathcal{O}(n^2)$ attention + $\mathcal{O}(n)$ SSM | Hybrid attention and SSM          | Decoding     | Combines selective global attention with linear-time sequence processing | Reduced direct access to earlier tokens in SSM layers         |
| H2O                | No asymptotic change; bounded KV cache              | Inference-time cache optimisation | Decoding     | Retains recent and high-attention tokens                                 | Eviction can remove information needed later                  |
| SparQ              | No asymptotic change; reduced cache transfer        | Approximate cache access          | Decoding     | Reduces memory bandwidth requirements                                    | Approximation can affect attention accuracy                   |

The approaches in Table 4 address different sources of the long-context bottleneck. Approximation-based methods such as Linformer and Reformer reduce the amount of attention computation by replacing exhaustive attention with a lower-dimensional or selectively sampled representation, primarily easing the prefilling phase. Longformer reduces computation by imposing a structured attention pattern that similarly targets prefilling cost. FlashAttention takes a different approach: it preserves the exact attention computation and improves practical efficiency by reducing memory traffic, with benefits that are most pronounced during prefilling. Mamba-based architectures such as Jamba do not approximate attention directly; instead, they replace many attention layers with a linear-time State Space mechanism and retain a smaller number of attention layers, principally easing the memory-bandwidth burden of decoding. Finally, H2O and SparQ target the inference-time KV cache directly during decoding, rather than the asymptotic complexity of the model architecture itself.

These distinctions show that long-context efficiency is not a single optimisation problem. Reducing arithmetic complexity, reducing memory traffic, reducing KV-cache size, and replacing the underlying sequence-processing mechanism are separate design choices, and each is typically most effective in one of the two inference phases identified in Section 2.5. A method that performs well for one bottleneck may provide limited benefit for another. Consequently, the appropriate strategy depends on whether the target workload is constrained primarily by prefilling computation, decoding memory bandwidth, accelerator memory capacity, or the need for direct access to distant information.

## 6.2 Empirical Time and Accuracy Comparison

Beyond the theoretical complexity summarised in Table 4, Table 3 consolidates representative empirical efficiency and quality results reported in the preceding approach sections. The results illustrate that long-context efficiency can be obtained through different mechanisms, including reduced attention computation, structured sparsity, hardware-aware execution, architectural changes, and KV-cache optimisation.

The reported quality effects are task-dependent. Linformer and Reformer achieve near-parity with their respective baselines in the reported evaluations, while Longformer reports improved results on the character-modelling and long-context tasks considered. FlashAttention preserves exact attention while improving execution efficiency. Jamba, H2O, and SparQ report substantial system-level efficiency gains under the long-context or KV-cache conditions evaluated in their respective studies.

These results are not a controlled head-to-head comparison. The methods were evaluated using different models, benchmarks, hardware configurations, context lengths, and workload regimes. Table 3 therefore summarises representative reported results rather than providing a common quantitative comparison.

## 6.3 Characteristic Limitations

The surveyed approaches introduce different limitations that become relevant under different workload requirements.

**Approximation error.** Approximation-based methods such as Linformer and Reformer reduce the number of operations by replacing exhaustive attention with a lower-dimensional or selectively sampled representation. Their effectiveness therefore depends on whether the approximation preserves the interactions required by the target task. Relevant information may be attenuated or omitted when the assumptions underlying the approximation do not hold.

**Restricted connectivity.** Structural sparsity methods such as Longformer deliberately restrict which token pairs can interact. Distant information can still propagate through multiple layers or through global tokens, but the model no longer has unrestricted direct access between every pair of positions. The choice of attention pattern therefore becomes an important part of model design.

**State-compression trade-off.** SSM-based architectures replace explicit token-to-token attention with a recurrent state representation. This provides favourable sequence-length scaling and avoids an attention-style growing KV cache, but it also means that information from previous positions is represented through a compressed state. Hybrid architectures such as Jamba retain attention layers to complement this limitation when direct token interactions are beneficial.

## 7 Conclusion

This report examined how the quadratic computational cost of standard self-attention, together with the quadratic memory required to materialise the attention matrix and the growing key-value cache during autoregressive inference, becomes increasingly restrictive as context lengths grow. The analysis addressed the research question of how modern attention and sequence-modelling strategies reduce these costs and how their respective trade-offs affect their suitability for long-context deployment.

Three main conclusions emerge from the comparison. First, different methods address different aspects of the long-context bottleneck, and different phases of inference. Linformer and Reformer reduce the amount of attention computation through approximation, primarily easing prefilling, while Longformer reduces computation by restricting the attention pattern, similarly targeting prefilling cost. These approaches can substantially improve scaling, but their effectiveness depends on assumptions about which token interactions are important.

Second, reducing algorithmic complexity is not the only route to practical efficiency. FlashAttention preserves the exact attention computation while reducing memory traffic and improving hardware utilisation. It therefore demonstrates that systems-level optimisation can provide substantial practical benefits even when the underlying  $\mathcal{O}(n^2)$  complexity remains unchanged.

Third, alternative sequence architectures and inference-time memory techniques address complementary constraints, particularly during decoding. Mamba provides linear-time sequence processing through a State Space representation, while Jamba combines Mamba layers with a smaller number of attention layers to retain direct token interactions where they are useful. H2O and SparQ demonstrate that the KV cache can also be treated as an independent optimisation target during inference. These approaches suggest that long-context deployment requires a combination of algorithmic, architectural, and systems-level design decisions, matched to whichever inference phase constrains the target workload.

**Limitations of this report.** This report is a literature-based survey and comparative analysis; it does not include original experiments or a common quantitative benchmark across the surveyed methods. The empirical figures reported in Sections 3–5 and the empirical comparison in Table 3 are drawn from the original publications’ own evaluation settings and therefore should not be interpreted as a controlled, like-for-like ranking. The Llama 3 case study in Table 2 is an illustrative calculation based on reported architectural parameters and the cache-memory equation, rather than a vendor-certified measurement. Quantitative conclusions for a particular deployment scenario should therefore be based on controlled experiments using the relevant model, hardware, context lengths, and workload.

**Outlook.** Important open challenges include developing methods that preserve precise long-range information access while reducing computational and memory requirements, adapting efficiency mechanisms to variable context lengths, and improving their interaction with accelerator memory hierarchies. Progress is likely to require joint consideration of algorithmic complexity, memory management, model architecture, and hardware execution rather than optimisation of any single component in isolation.

# References

- [BPC20] Iz Beltagy, Matthew E. Peters, and Arman Cohan. “Longformer: The Long-Document Transformer”. In: *arXiv preprint arXiv:2004.05150* (2020).
- [Dao+22] Tri Dao et al. “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness”. In: *Advances in Neural Information Processing Systems*. Vol. 35. 2022, pp. 16344–16359.
- [Dev+19] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *arXiv preprint arXiv:1810.04805* (2019).
- [GD23] Albert Gu and Tri Dao. “Mamba: Linear-Time Sequence Modeling with Selective State Spaces”. In: *arXiv preprint arXiv:2312.00752* (2023).
- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780.
- [KKL20] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. “Reformer: The Efficient Transformer”. In: *International Conference on Learning Representations*. 2020.
- [Lie+24] Opher Lieber et al. “Jamba: A Hybrid Transformer-Mamba Language Model”. In: *arXiv preprint arXiv:2403.19887* (2024).
- [Lla24] Llama Team. “The Llama 3 Herd of Models”. In: *arXiv preprint arXiv:2407.21783* (2024).
- [Rib+24] Luka Ribar et al. “SparQ Attention: Bandwidth-Efficient LLM Inference”. In: *Proceedings of the 41st International Conference on Machine Learning*. Vol. 235. 2024, pp. 42558–42583.
- [Vas+17] Ashish Vaswani et al. “Attention Is All You Need”. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.
- [Wan+20] Sinong Wang et al. “Linformer: Self-Attention with Linear Complexity”. In: *arXiv preprint arXiv:2006.04768* (2020).
- [Zha+23] Zhenyu Zhang et al. “H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models”. In: *Advances in Neural Information Processing Systems*. Vol. 36. 2023.

## A Work Sharing

This report was completed individually by the author. The work-sharing section is therefore not applicable.

## B Code Samples

This report is a literature-based survey and comparative analysis; no original software was implemented and no code samples were produced as part of this work. Equations and algorithmic descriptions presented in the report are derived from the cited publications rather than from an original implementation.

## C Transparency Report

See the Declaration on the Use of AI and Third-Party Assistance immediately following the title page.