computers

MDPI

Article

Computational Complexity of Self-Attention Algorithms for Efficient Transformer Architectures

Akinyemi Omololu Akinrotimi * and Sunday Olusegun Ojo

Faculty of Accounting and Information Technology, Durban University of Technology, Durban 4001, South Africa; sundayo1@dut.ac.za

* Correspondence: akinrotimiao@dut.ac.za

Abstract

Self-attention is fundamental to Transformer architectures, but its computational and memory requirements grow quadratically with sequence length. This paper presents a structured narrative review of computationally efficient self-attention algorithms. It establishes the mathematical foundations of standard self-attention and examines dense, sparse/local, low-rank projection, landmark/Nyström, kernel/linear, memory-efficient exact, and hierarchical mechanisms. The review compares these approaches in terms of arithmetic complexity, memory requirements, sequence-length scaling, approximation, connectivity, I/O behaviour, implementation overhead, and hardware dependence. It distinguishes algorithmic complexity reduction from implementation-level optimization and clarifies the conditions under which complexity is linear in sequence length. Analytical crossover conditions show how sequence length, representation dimension, mechanism-specific parameters, and implementation overhead influence practical advantage. The review demonstrates that efficient attention cannot be characterized solely as quadratic versus linear complexity and identifies research gaps in multidimensional complexity modelling, approximation-error trade-offs, I/O complexity, long-range dependency preservation, Pareto efficiency, and hardware-aware analysis.

Keywords: self-attention; transformer architectures; computational complexity; efficient attention; low-rank attention; hardware-aware efficiency

1. Introduction

Transformer architectures have become a widely used framework for sequence modelling because self-attention provides direct interactions among elements of an input sequence and allows representations to incorporate information from distant positions [1]. The original Transformer formulation established scaled dot-product self-attention as a central computational mechanism, replacing recurrent processing with attention-based parallel computation [1]. Subsequent Transformer architectures have demonstrated the applicability of this formulation across natural language processing, computer vision, multimodal learning, and other domains [2]. As sequence lengths and model dimensions continue to increase, however, the computational and memory requirements associated with attention have become increasingly important considerations in the design and deployment of Transformer-based systems [2].

For an input sequence of length n and representation dimension d, conventional self-attention explicitly computes interactions between pairs of sequence elements. The resulting attention-score computation has a dominant arithmetic complexity of $O(n^2d)$,

Check for updates

Academic Editor: Martin Valtierra-Rodriguez

Received: 9 October 2026
Revised: 28 October 2026
Accepted: 30 October 2026
Published: 10 November 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

while explicit storage of the dense attention matrix requires $O(n^2)$ memory [1,2]. The linear projections used to construct the query, key, and value representations contribute an additional $O(nd^2)$ cost under dense d-to-d projections [1]. Consequently, the leading-order computational cost associated with the attention block can be expressed as $O(nd^2 + n^2d)$. The quadratic dependence on sequence length becomes increasingly significant as n grows, particularly in long-context applications where processing sequences substantially longer than those used in conventional Transformer settings is required [1,2].

The quadratic dependence on sequence length has therefore become one of the principal computational challenges associated with Transformer architectures [1,2]. Increasing the context length from n to $2n$, for example, can increase the size of the pairwise attention computation by approximately four times. This characteristic creates substantial demands on processing units, memory capacity, data movement, and energy consumption [2,3]. The problem is not limited to the theoretical operation count. In practical implementations, the storage and movement of intermediate attention matrices can become important bottlenecks, particularly on modern accelerators where memory bandwidth and on-chip storage can constrain performance [3].

The computational limitations of dense self-attention have motivated diverse approaches that reduce token interactions, modify attention representations or mathematical formulations, and improve computational execution, with sparse attention representing an early approach to reducing the cost of token interactions [2,4]. Sparse and local mechanisms restrict explicit interactions to selected positions, thereby reducing the number of pairwise computations [5–7]. Low-rank approaches approximate the attention computation through reduced-dimensional projections or representations [8], whereas kernel- and feature-based approaches reformulate attention to avoid explicitly constructing the complete pairwise attention matrix [9,10]. Landmark-based methods, including Nyström-based attention, approximate the attention structure using a smaller set of representative positions and therefore require separate consideration from projection-based low-rank methods [7,8]. Other approaches improve efficiency without changing the underlying asymptotic arithmetic complexity. FlashAttention, for example, retains the exact attention computation while reorganizing its execution to reduce memory traffic and improve utilization of modern accelerator architectures [3]. Hierarchical and multi-scale approaches provide additional mechanisms for controlling the cost of attention by operating at different levels of representation or resolution.

The resulting literature therefore contains several fundamentally different meanings of "efficient attention" [2]. A method may reduce computational complexity by limiting the number of token interactions, reducing the dimensionality of the representation, reformulating the attention computation, or exploiting a more efficient computational execution strategy [3,5,8,9,11–14]. These mechanisms should not be treated as equivalent solely because they produce a lower computational cost under particular assumptions. For example, sparse attention mechanisms modify the interaction pattern, as illustrated by locality-sensitive hashing in Reformer and shifted-window attention in Swin Transformer [12,13]. Low-rank methods modify the representation used to approximate attention, while linear-complexity approaches such as Efficient Attention reformulate the computation to reduce its complexity [14]. An IO-aware implementation, by contrast, can improve execution efficiency while preserving the exact attention result [3].

Existing surveys have provided important overviews of efficient Transformer models and classified methods according to mechanisms such as sparse attention, low-rank approximation, kernelization, recurrence, and other forms of approximation [2,15]. However, comparisons across these approaches are complicated by differences in the quantities reported and the assumptions under which complexity is derived [2,5,8–10]. Some studies

emphasize asymptotic arithmetic complexity, whereas others focus on memory consumption, approximation quality, sequence-length scalability, or implementation efficiency [2–4]. In addition, a complexity expression that is linear in sequence length generally assumes that other parameters, such as the representation dimension, local window size, reduced rank, or feature dimension, are fixed with respect to n [2,8,9]. Consequently, asymptotic notation alone does not determine the sequence length at which an efficient mechanism becomes practically advantageous. Constant factors, memory access patterns, hardware characteristics, and implementation overhead can materially affect this relationship [3,4].

The distinction sought in this review is analytical rather than taxonomic. Existing surveys establish the principal families of efficient Transformer architectures, but a classification by algorithm family does not, by itself, establish whether reported complexity expressions refer to attention interactions alone or to a broader attention block, which parameters must remain fixed for a linear or subquadratic scaling claim, or when an asymptotic reduction may outweigh additional operations and implementation overhead. The present review addresses these questions by relating each efficiency mechanism to its dominant computational operations, relevant control parameters, memory behaviour, approximation or connectivity assumptions, and implementation constraints. Its contribution is therefore a common basis for interpreting and comparing complexity claims, including the conditions and limitations under which those claims hold, rather than the proposal of a new algorithm or a claim that earlier surveys have overlooked efficiency altogether. These observations motivate a comparative treatment in which self-attention mechanisms are examined using a consistent set of computational dimensions. In this review, the literature is organized according to the principal mechanism through which computational or memory costs are reduced. The analysis considers time complexity, memory requirements, sequence-length scaling, exactness versus approximation, attention-matrix construction, algorithm-specific control parameters, connectivity characteristics, and practical implementation considerations. Particular attention is given to the distinction between theoretical arithmetic complexity and practical execution efficiency, since a reduction in asymptotic growth does not necessarily imply a proportional reduction in wall-clock execution time or hardware resource consumption [3,4].

The review also examines the conditions under which different complexity regimes become relevant. Rather than treating expressions such as $O(n^2d)$, as $O(\mathrm{nw}d)$, and as $O(\mathrm{nr}d)$ as direct measures of execution time, the analysis considers the parameters and assumptions that determine their relative computational costs [2,3]. This provides a basis for examining crossover conditions between dense and reduced-complexity attention and for identifying the computational regimes in which different strategies may become advantageous. The review further considers approximation and connectivity as complementary dimensions of efficiency, recognizing that reducing computation may involve changes in the ability to represent unrestricted pairwise relationships [5–7]. Accordingly, the main contribution of this review is not the introduction of a new self-attention mechanism or a new empirical benchmark. Rather, it is a structured synthesis and comparative interpretation of established results concerning the computational complexity of self-attention mechanisms. The review provides a unified analytical perspective by relating computational cost, memory requirements, sequence-length scaling, approximation characteristics, connectivity, and implementation considerations across major classes of attention mechanisms. It also identifies unresolved issues and formulates specific research questions concerning crossover behavior, approximation-error analysis, memory and I/O costs, long-range dependency preservation, and multi-objective efficiency trade-offs.

The remainder of the paper is organized as follows. Section 2 describes the review methodology and scope, including the literature-search and selection procedures. Section 3

establishes the mathematical and computational foundations required for the complexity analysis. Section 4 develops the taxonomy of self-attention mechanisms according to their principal efficiency strategies. Section 5 analyzes their computational and memory complexity. Section 6 discusses practical execution and hardware considerations. Section 7 identifies the research gaps and specific open research questions arising from the computational-complexity perspective. Section 8 discusses the broader implications of the review, Section 9 highlights the limitations of the study and Section 10 concludes the paper.

2. Review Methodology

2.1. Review Type and Scope

This study is a structured narrative review of the computational complexity of self-attention algorithms for efficient Transformer architectures. It is not intended as a systematic literature review or meta-analysis. The objective is to synthesize and theoretically compare representative self-attention mechanisms according to the ways in which they affect computational cost, memory requirements, sequence-length scaling, approximation, connectivity, and implementation efficiency.

The review focuses on the computational mechanisms underlying efficient self-attention rather than on application-specific predictive performance. Foundational attention mechanisms are considered alongside subsequent approaches that seek to reduce, restructure, or better manage the computational and memory requirements of attention [1,2]. Recent developments are also considered where they introduce computationally relevant modifications to attention or provide contemporary perspectives on efficiency-oriented Transformer design, including recent surveys that examine hardware-efficient, sparse, compact, and linear attention methods [4,5,16,17].

The narrative-review approach is adopted because efficient self-attention encompasses mechanisms with substantially different computational principles [2], and narrative reviews are particularly suited to broad or heterogeneous topics that require synthesis, interpretation, and critical comparison rather than an exhaustive synthesis of narrowly defined studies [18]. Restricting interactions, reducing representation dimensionality, reformulating the attention operation, approximating the attention structure, and reorganizing exact computation for improved memory utilization represent different forms of efficiency and therefore require comparative theoretical analysis rather than treatment as a single homogeneous class [2,3].

2.2. Literature Identification and Selection

The literature was identified through targeted searches of scholarly databases and digital libraries, supplemented by examination of the reference lists of foundational papers and relevant review articles. The search focused on research addressing the computational complexity, memory requirements, sequence-length scalability, approximation behaviour, and implementation of self-attention in Transformer architectures. The literature selection included foundational work on computational complexity and attention mechanisms, beginning with the development of the Transformer architecture in 2017, alongside earlier foundational contributions to computational complexity where relevant. The review also considered subsequent research on sparse, linear, low-rank, landmark-based, and memory-efficient attention mechanisms, including studies published through 2025. The sources were selected for their relevance to the theoretical analysis and comparison of computational costs rather than solely for application-specific performance.

Studies were considered relevant when they contributed to the mathematical formulation, computational complexity, memory requirements, approximation behaviour, interaction structure, or implementation of self-attention mechanisms in Transformer ar-

chitectures. Foundational papers were retained where necessary to explain the origin of an algorithm or its computational formulation, while more recent publications were considered to reflect subsequent developments. Publications were excluded when they did not provide a substantive connection to self-attention computation, offered insufficient information for the intended theoretical comparison, or focused on application outcomes without informing the computational mechanisms under review.

Selection proceeded by examining titles and abstracts for relevance, followed by closer inspection of the full text of potentially relevant publications. The selected studies were then compared according to the computational mechanism used, the dominant operations involved, the assumptions underlying reported complexity, memory behaviour, exactness or approximation, and implementation considerations. The resulting collection was assessed for coverage across the principal attention families and for inclusion of both foundational methods and subsequent developments. This process was intended to support a technically representative narrative synthesis rather than an exhaustive systematic review or a statistical aggregation of results.

2.3. Taxonomy of Representative Attention Mechanisms

The selected mechanisms are organized according to the principal computational strategy through which attention efficiency is obtained or managed. The review distinguishes:

(i) dense or full self-attention, which explicitly computes unrestricted pairwise interactions [1];
(ii) sparse and local attention, which reduces the number of explicit interactions through restricted or structured connectivity [4–6];
(iii) low-rank projection-based attention, which reduces the effective representation used in attention computation through projection or factorization [7];
(iv) landmark or Nyström-based attention, which approximates the attention structure using a smaller set of representative positions [8];
(v) kernel or linear attention, which reformulates the attention operation to avoid explicit construction of the complete pairwise attention matrix [9,10];
(vi) memory- and hardware-efficient exact attention, which retains the exact attention computation while reorganizing its execution to reduce intermediate memory requirements or improve memory movement and hardware utilization [3,11]; and
(vii) hierarchical or multi-scale attention, which controls computational requirements by operating on representations at different spatial, temporal, or semantic scales [12].

A summary of representative self-attention mechanisms and their computational characteristics is provided in Table 1. The separation of landmark or Nyström-based attention from low-rank projection-based attention is deliberate. Although both can reduce the computational burden associated with dense attention, they employ different approximation constructions [7,8]. Projection-based methods reduce or transform the representation used in the attention calculation, whereas landmark-based methods use representative positions to approximate the attention structure [7,8]. Their computational complexity, approximation assumptions, and sources of error therefore warrant separate consideration.

Table 1. Self-Attention Mechanisms and Their Computational Characteristics.

Attention Mechanism	Category	Main Efficiency Principle	Exact/ Approximate	Primary Complexity Characteristic
Self-attention [1]	Dense attention	Full pairwise interaction	Exact	$O(nd^2)$, attention computation; $O(n^2)$, attention-matrix storage
Sparse attention [4]	Sparse attention	Restrict token interactions	Exact with respect to defined pattern	Approximately $O(n$wd),

Table 1. *Cont.*

Attention Mechanism	Category	Main Efficiency Principle	Exact/ Approximate	Primary Complexity Characteristic
Longformer [5]	Local/global sparse attention	Local windows with selected global positions	Exact for defined pattern	Approximately linear in n for fixed window/global connectivity
BigBird [6]	Structured sparse attention	Local, global, and random interactions	Exact for defined pattern	Approximately linear in n for fixed sparse connectivity
Linformer [7]	Low-rank attention	Project keys and values into lower-dimensional representations	Approximate	$O(nrd)$ under reduced rank r
Nyströmformer [8]	Landmark-based attention	Approximate attention using landmark points	Approximate	Depends on n, landmark count m, and d
Performer [10]	Kernel-based attention	Kernel feature-map reformulation	Approximate	Linear in n for fixed feature dimension p
Reformer [13]	Locality-sensitive hashing	Hash-based restriction of attention comparisons	Approximate/ structured	$O(n\log n)$ attention complexity under LSH formulation
FlashAttention [3]	Memory -efficient exact attention	Tiling and I/O-aware computation	Exact	Quadratic arithmetic; reduced memory/I/O overhead
FlashAttention-2 [11]	Memory-efficient exact attention	Improved parallelism and work partitioning	Exact	Quadratic arithmetic; improved parallel and memory efficiency
FlashAttention-3 [19]	Hardware-aware, memory-efficient exact attention	Asynchronous GPU execution, warp specialization, overlapping computation and data movement, and low-precision FP8 processing	Exact attention algorithm; FP8 uses reduced precision and may introduce numerical error	Quadratic arithmetic in sequence length n, with improved GPU utilization and reduced memory-traffic overhead
Swin Transformer [12]	Hierarchical/window attention	Windowed and shifted-window interactions	Exact for defined windows	Linear in image-token count for fixed window size

2.4. Analytical Dimensions

The attention mechanisms are compared using a common set of theoretical and computational dimensions. Let n denote sequence length and d denote representation dimension. Where necessary, additional algorithm-specific parameters are introduced, including local window size w, reduced rank r, feature dimension p, and number of landmarks m.

The analysis considers:

(i) time complexity, expressed in terms of the dominant arithmetic operations;
(ii) memory complexity, including attention-related intermediate storage;
(iii) attention-matrix construction, particularly whether a complete $n \times n$ matrix is explicitly materialized;
(iv) exactness versus approximation, distinguishing mechanisms that preserve exact attention from those that approximate it;
(v) algorithm-specific parameters, including local window size w, reduced rank r feature dimension p, and number of landmarks m, which determine computational cost under particular configurations;

(vi) connectivity characteristics, including unrestricted, sparse, local, sampled, or hierarchical interactions; and
(vii) implementation dependence, including memory movement, intermediate storage, and hardware utilization.

This study also distinguishes complexity that is linear in sequence length from total computational complexity. For example, expressions such as $O(nwd)$,$O(nrd)$, and $O(npd)$ are linear in n only when d,w,r, and p, respectively, are treated as fixed with respect to sequence length. Consequently, a lower asymptotic order does not by itself establish that an efficient mechanism will be faster for every combination of sequence length, representation dimension, and algorithm-specific parameters.

2.5. Scope and Analytical Limitations

The review is concerned primarily with the computational properties of self-attention mechanisms rather than with establishing the predictive superiority of any particular Transformer architecture. No new dataset, model-training experiment, synthetic-data experiment, or hardware benchmark is introduced. Empirical results reported by the original studies are considered only as supporting evidence concerning the practical behaviour of the corresponding mechanisms.

The analysis therefore distinguishes theoretical complexity from measured execution performance. Big-O expressions describe asymptotic growth and do not directly specify wall-clock execution time [2,3]. Constant factors, memory bandwidth, kernel implementation, hardware architecture, parallelism, and other implementation characteristics can affect the practical relationship between theoretically derived complexity and observed execution [3,11].

Similarly, the classification adopted in this review is based on underlying computational mechanisms rather than terminology alone. Terms such as efficient attention, linear attention, sparse attention, and memory-efficient attention are not assumed to describe equivalent computational properties. The taxonomy is consequently intended to provide a consistent analytical basis for comparing methods that may use different terminology in the original literature.

On this basis, the subsequent analysis examines the mathematical foundations of self-attention, develops the proposed taxonomy, compares the computational and memory characteristics of representative mechanisms, and investigates the conditions under which different complexity regimes may become advantageous.

3. Mathematical Foundations of Self-Attention

3.1. Input Representation and Linear Projections

Let the input sequence be represented by a matrix $X \in \mathbb{R}^{n \times d}$, where n denotes the sequence length and d denotes the representation dimension. Standard self-attention transforms X into query, key, and value representations through learned linear projections [1]:

$$Q = XW_Q,\ K = XW_K,\ V = XW_V, \tag{1}$$

where $W_Q, W_K, W_V \in \mathbb{R}^{d \times d}$. Under these dimensions, each projection requires a dense matrix multiplication between an $n \times d$ matrix and a $d \times d$ matrix. Consequently, the arithmetic cost of each projection is

$$O\left(nd^2\right)$$

The three projections therefore have a combined arithmetic cost of

$$3nd^2 = O\left(nd^2\right), \tag{2}$$

since the constant factor of three is suppressed in asymptotic notation.

This distinction is important for the subsequent complexity analysis. The projection operation is linear in sequence length n when d is treated as fixed, but its cost remains quadratic in the representation dimension d. Thus, referring to the projection cost simply as "linear complexity" without specifying the relevant variable can be misleading.

3.2. Scaled Dot-Product Self-Attention

The standard scaled dot-product self-attention mechanism first computes pairwise query-key similarities [1]:

$$S = \frac{QK^T}{\sqrt{d_k}} \tag{3}$$

where $Q, K \in \mathbb{R}^{n \times d}$. Here, (Q), (K), and (V) denote the query, key, and value matrices, respectively, and d_k is the dimension of the query and key vectors for the attention head. The scaling factor is therefore defined using the head-specific query/key dimension. Where the value dimension differs from the key dimension, the dimensions of the final matrix multiplication must be stated accordingly. Because Q and $K^\top$ have dimensions $n \times d$ and $d \times n$, respectively, the multiplication $QK^\top$ produces an $n \times n$ matrix and requires $O(n^2 d)$ arithmetic operations. The attention weights are then obtained by applying the row-wise softmax operation:

$$A = \text{softmax}(S). \tag{4}$$

where $A \in \mathbb{R}^{n \times n}$. The subsequent multiplication of the attention matrix by the value representation is

$$Y = AV. \tag{5}$$

where Y = Attention (Q, K, V). Since $A \in \mathbb{R}^{n \times n}$ and $V \in \mathbb{R}^{n \times d}$, this operation also requires $O(n^2 d)$ arithmetic operations. The complete scaled dot-product attention operation can therefore be written as

$$Attention\ (Q,\ K,\ V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{6}$$

Equations (3)–(6) separate the principal computational stages of self-attention rather than combining them into a single undifferentiated complexity expression. This separation is useful when comparing efficient attention mechanisms because different approaches reduce or reorganize different components of the computation. The pairwise attention computation has a dominant arithmetic cost of $O(n^2 d)$ when the total query, key, and value dimensions are proportional to the model dimension (d). The complete multi-head attention block also includes the linear projections that construct the query, key, and value representations, as well as project the concatenated head outputs. Under conventional dense projections, these operations contribute terms of order $O(n^2 d)$.

3.3. Overall Computational Complexity

Combining the projection operations with the two dominant pairwise matrix multiplications gives the leading-order computational complexity of standard self-attention as [1,2]

$$C_{dense}\ (n, d) = an^2 d + bnd^2 \tag{7}$$

where a and b represent operation-count coefficients determined by the adopted counting convention and implementation. The quadratic term describes pairwise attention, whereas the second term represents projection work.

3.4. Memory Complexity

The score matrix S and the normalized attention matrix A both have dimensions $n \times n$. Explicit materialization of either matrix therefore requires

$$O\left(n^2\right)$$

memory. The quadratic memory requirement is distinct from the arithmetic complexity of attention. A method may retain $O(n^2d)$ arithmetic complexity while avoiding explicit storage of an $n \times n$ intermediate matrix, or it may reduce the number of computed interactions while still using a dense representation internally. Consequently, time complexity and memory complexity must be analysed separately [2,3].

For standard attention, the $n \times n$ attention structure is particularly significant as sequence length increases. Doubling n, for example, increases the number of pairwise positions by a factor of four. This quadratic growth is one of the principal computational constraints motivating sparse, approximate, reformulated, and memory-efficient attention mechanisms [2,4–11].

3.5. Distinguishing Projection, Interaction, and Memory Costs

For clarity, the computational components of standard self-attention can be summarized in Table 2 as follows:

Table 2. Summary of Computational Components of Standard Self-Attention.

Component	Operation	Leading-Order Complexity
Query, key, and value projections	$Q = XW_Q, K = XW_K, V = XW_V$	$O(nd^2)$
Query-key interaction	$S = QK^\top/\sqrt{d}$	$O(n^2d)$
Attention normalization	softmax (S)	$O(n^2)$
Attention-value interaction	$Y = AV$	$O(n^2d)$
Combined leading-order computation	Projection + attention	$O(nd^2 + n^2d)$
Explicit attention storage	Storage of S and/or A	$O(n^2)$

Table 2 emphasizes an important distinction for the remainder of the review. The quadratic dependence on sequence length originates primarily from the pairwise interaction structure, whereas the projection operations introduce a separate $O(nd^2)$ computational component. Efficient attention mechanisms can therefore be understood partly according to which of these computational or memory components they modify [2].

The computational and memory characteristics of the self-attention mechanisms considered in this study are summarized in Table 3. The comparison focuses primarily on the dominant computational and memory terms associated with each mechanism, while distinguishing exact attention from approaches that obtain efficiency through sparsity, dimensionality reduction, approximation, or computational reformulation. The expressions are stated under the parameter assumptions described in the preceding analysis; in particular, parameters such as the local window size w, reduced rank r, and feature dimension p are treated as fixed with respect to sequence length n when discussing linear dependence on n. Landmark-based methods are considered separately because their complexity also depends on the number of landmarks m. The table is intended to provide a theoretical basis for comparing the different mechanisms rather than to imply that asymptotic complexity alone determines practical execution performance. Factors such as memory movement,

implementation overhead, hardware utilization, and approximation error are considered separately in the subsequent analysis.

Table 3. Comparison of Self-Attention Mechanisms by Computational and Memory Complexity.

Mechanism	Dominant Computational Complexity	Attention-Related Memory Complexity	Dependence on Sequence Length	Exactness
Self-attention	$O(nd^2 + n^2d)$	$O(n^2)$	Quadratic	Exact
Local/ window attention	$O(nd^2 + nwd)$	$O(nw)$	Linear for fixed w	Exact within window
Sparse attention	Pattern-dependent	Pattern-dependent	Subquadratic or linear depending on pattern	Exact within pattern
Linformer	$O(nd^2 + nrd)$	Approximately $O(nr)$	Linear for fixed r	Approximate
Performer	Approximately $O(nd^2 + npd)$	Approximately $O(np)$	Linear for fixed p	Approximate
Nyströmformer	Depends on landmark count m	Depends on m and implementation	Reduced dependence on n for fixed m	Approximate
Reformer	Pattern/hash dependent	Reduced relative to dense attention	Sub-quadratic /approximately linear under assumptions	Approximate
FlashAttention	$O(nd^2 + n^2d)$	$O(nd)$ working memory under tiled implementation	Quadratic arithmetic	Exact
FlashAttention-2	$O(nd^2 + n^2d)$	$O(nd)$ working memory under tiled implementation	Quadratic arithmetic	Exact
FlashAttention-3	$O(nd^2 + n^2d)$	$O(nd)$ working memory under tiled implementation	Quadratic arithmetic; improved GPU execution through asynchronous operations and computation–data movement overlap	Exact attention; low-precision implementations may introduce numerical error
Swin Transformer	$O(nd^2 + nwd)$ for fixed windows	Window-dependent	Linear for fixed window size	Exact within windows

Asymptotically, the interaction cost of the reduced-complexity method grows more slowly with sequence length than that of dense attention when its reduction parameters remain fixed. This asymptotic relationship does not establish a lower numerical cost for every finite sequence length. A finite-length comparison requires explicit cost functions that account for the leading operations and any method-specific overhead.

3.6. Interpretation of Sequence-Length Scaling

The expression $O(nd^2 + n^2d)$ should not be interpreted as meaning that every component of self-attention is quadratic. The projection component is linear in n for fixed d, while the pairwise interaction and attention-value multiplication exhibit quadratic dependence on n. The distinction becomes particularly important when comparing mechanisms described as "linear attention".

For example, a complexity expression of the form $O(nrd)$ is linear in n only under the assumption that the reduced dimension r and representation dimension d do not grow with n. Likewise, $O(nwd)$ is linear in n under a fixed local-window size w For kernel or feature-map approaches such as Performer, an expression involving p, the feature-map dimension, is similarly linear in n only when p and d are treated as fixed with respect to n. These assumptions are therefore stated explicitly whenever the review characterizes an attention mechanism as linear in sequence length [2,7,10].

The theoretical complexity expressions provide the mathematical basis for the comparative analysis that follows. However, they do not by themselves determine practical execution time. Constant factors, memory access, parallelization, hardware characteristics, implementation strategy, and approximation overhead can affect the point at which an alternative mechanism becomes advantageous [3,11]. These considerations are examined later in the review rather than being inferred directly from asymptotic notation.

4. Taxonomy of Efficient Self-Attention Mechanisms

The computational formulation of standard self-attention provides a basis for understanding why alternative attention mechanisms have been developed. The principal approaches do not reduce computational cost in the same manner. Some reduce the number of token-to-token interactions, some approximate the attention structure, some reduce the dimensionality of the computation, and others preserve the exact attention result while improving memory or hardware efficiency. A meaningful complexity comparison therefore requires these mechanisms to be distinguished according to their underlying computational principles.

This review organizes representative efficient self-attention mechanisms into seven classes: sparse and local attention, low-rank projection-based attention, landmark-based attention, kernel or linear attention, memory- and hardware-efficient exact attention, and hierarchical attention, with dense self-attention serving as the reference formulation. The classification is based on the computational transformation applied to the attention operation rather than on the terminology adopted by individual studies, consistent with the need for mechanism-level distinctions in contemporary efficient-attention taxonomies [17]. In particular, landmark-based methods such as Nyströmformer are treated separately from low-rank projection-based methods such as Linformer because their computational constructions and controlling parameters differ [7,8].

4.1. Dense Self-Attention as the Reference Mechanism

Dense self-attention permits every element in a sequence to interact with every other element. Consequently, an input sequence of length n produces n^2 potential pairwise interactions. As established in Section 3, the leading-order computational complexity is $O(nd^2 + n^2d)$, while explicit storage of the attention matrix requires $O(n^2)$ memory [1,2].

Dense self-attention provides unrestricted global connectivity and does not introduce an approximation to the attention matrix. These properties make it a useful reference against which computationally efficient alternatives can be examined.

The principal limitation is the quadratic dependence on sequence length. As n increases, the pairwise interaction component becomes increasingly costly in both computa-

tion and memory. Efficient attention mechanisms can therefore be viewed as attempts to modify one or more of the following properties: the number of interactions, the representation dimension involved in those interactions, the manner in which attention is computed, or the manner in which intermediate results are stored.

4.2. Sparse and Local Attention

Sparse attention reduces computational requirements by restricting the set of positions that each token can attend to. Instead of computing all n^2 pairwise interactions, each position interacts with only a subset of the sequence. The subset may be determined by local windows, fixed patterns, blocks, dilated structures, or other predefined or learned connectivity patterns [4–6].

For a fixed local window of size w, the number of attention interactions is approximately $O(nw)$. The corresponding attention interaction cost can therefore be expressed as $O(nwd)$, under the assumption that w is independent of n.

The principal advantage is the reduction from quadratic dependence on sequence length to a dependence that is linear in n under a fixed-window assumption. However, this reduction is accompanied by a change in connectivity. Unlike dense attention, a local mechanism does not provide direct interaction between every pair of sequence elements. Information from distant positions must instead propagate through the available interaction structure [4,5].

Sparse attention therefore represents more than an arithmetic optimization. The choice of sparsity pattern determines which relationships are directly represented and how efficiently information can propagate across the sequence [4–6]. Furthermore, the theoretical memory benefit depends on the implementation. A sparse mathematical pattern does not necessarily imply $O(nw)$ physical storage if a dense mask or dense intermediate representation is materialized [3].

4.3. Low-Rank Projection-Based Attention

Low-rank projection-based approaches reduce the computational burden of attention by projecting the key and value representations into a lower-dimensional space. Rather than performing the complete attention computation using the original sequence dimension, a reduced representation of dimension r is introduced, where r is substantially smaller than n under the intended operating conditions. Linformer is a representative example of this projection-based approach [7].

A representative complexity form for such mechanisms is $O(nrd)$, subject to the particular factorization and projection operations used by the algorithm. The computational benefit arises from replacing operations involving the full $n \times n$ interaction structure with operations involving a reduced representation. The effectiveness of the reduction therefore depends on the relationship between n,r, and d. In particular, describing the resulting complexity as linear in sequence length assumes that r and d do not increase proportionally with n.

The principal theoretical trade-off is between computational efficiency and the information retained by the reduced representation. A smaller r can reduce computational cost but may also constrain the representational capacity of the approximation. Consequently, the rank parameter should be considered an explicit component of complexity analysis rather than treated as an inconsequential implementation detail [7].

4.4. Landmark-Based and Nyström Attention

Landmark-based attention constitutes a distinct approximation strategy in which a smaller collection of representative sequence positions, referred to as landmarks, is used

to approximate the full attention structure. Nyström-based approaches are an important example of this class [8].

If the number of landmarks is denoted by m, where $m < n$, the full $n \times n$ attention structure can be approximated using interactions involving the selected landmarks rather than directly computing every pairwise interaction. The resulting computational behaviour depends on the precise approximation procedure and implementation, but its central principle is the replacement of the full attention structure with an approximation induced by representative positions [8].

Landmark-based attention should therefore not simply be absorbed into the broader low-rank category. Although the resulting approximation may have a low-rank interpretation, the mechanism by which that approximation is constructed is materially different from methods that directly project representations into a reduced-dimensional space, such as Linformer [7,8].

Accordingly, although Nyströmformer uses a low-rank approximation of the attention structure, its computational cost should not be represented by the same generic $O(nrd)$ expression used for projection-based low-rank methods. The relevant reduction parameter is the number of landmarks m, and the overall cost also depends on landmark selection, landmark-to-token interactions, and the approximation and reconstruction operations used by the specific Nyström formulation [8].

Three characteristics are particularly relevant to its analysis: the number and selection of landmarks, the approximation used to reconstruct the attention structure, and the resulting approximation error. Increasing the number of landmarks can improve the fidelity of the approximation while increasing computational requirements. Thus, the landmark count m represents an explicit efficiency-accuracy control parameter [8].

4.5. Kernel and Linear Attention

Kernel or linear attention approaches reformulate the attention computation so that sequence-dependent summations can be rearranged before the complete pairwise attention matrix is constructed. The principal objective is therefore to avoid explicitly computing and storing the full attention matrix [9,10,14].

Under an appropriate feature or kernel representation of dimension p, a representative complexity form is $O(npd)$, where p denotes the dimension of the feature or kernel representation used by the particular formulation. This expression is linear in sequence length n when p and d are treated as fixed with respect to n.

The important distinction from sparse attention is that kernel or linear attention does not necessarily achieve efficiency by removing particular token-to-token connections. Instead, it changes the algebraic organization of the computation. Consequently, the complete $n \times n$ attention matrix can be avoided while maintaining a form of global sequence interaction [9,10].

The computational advantage therefore depends on the feature representation and the dimensions involved. Moreover, the exact relationship to standard softmax attention depends on the formulation. Some linear-attention approaches exploit an algebraic reformulation of the attention computation, whereas approximation-based kernel methods, such as Performer, use feature representations to approximate the softmax attention operation [9,10]. The computational reduction should consequently be considered together with the assumptions, feature dimension, and approximation characteristics of the particular formulation.

4.6. Memory- and Hardware-Efficient Exact Attention

Not all efficient attention mechanisms reduce the asymptotic number of pairwise operations. Memory- and hardware-efficient exact attention methods instead reorganize the computation so that the exact attention result can be obtained with substantially reduced intermediate memory requirements and more efficient data movement [3,11].

FlashAttention provides a representative example. Its principal distinction from approximate attention mechanisms is that it does not replace the mathematical attention operation with a low-rank, sparse, or kernel approximation. Instead, the computation is organized around tiling, memory-aware execution, and improved use of the available memory hierarchy [3]. FlashAttention-2 extends this approach by improving parallelism and work partitioning while retaining exact attention computation [11]. FlashAttention-3 further advances hardware-aware execution by exploiting asynchronous operations, warp specialization, and overlap between computation and data movement on modern GPUs, while retaining the quadratic arithmetic complexity of full attention [19].

This distinction is important for the complexity taxonomy. An exact method can preserve the fundamental quadratic pairwise arithmetic while reducing the practical cost associated with storing and moving intermediate attention matrices. Thus, asymptotic arithmetic complexity and practical execution efficiency describe different dimensions of attention efficiency [3,11]. This class consequently demonstrates why a review based solely on Big-O arithmetic complexity would be incomplete. Two mechanisms can have comparable asymptotic arithmetic complexity while exhibiting substantially different memory traffic, intermediate storage requirements, and accelerator utilization [3,11].

4.7. Hierarchical and Multi-Scale Attention

Hierarchical attention reduces computational requirements by operating on representations at multiple levels of granularity. Rather than applying unrestricted attention over the complete sequence or spatial representation at a single resolution, the architecture can restrict detailed attention to smaller regions while using coarser representations to capture broader dependencies. A representative example is the hierarchical design of Swin Transformer, where local window-based interactions are combined with shifted windows across successive stages to facilitate information exchange between neighbouring regions and representational scales [12].

The computational advantage arises primarily from reducing the number of high-resolution pairwise interactions. However, the resulting complexity depends on the hierarchy, number of stages or scales, region sizes, token reduction strategy, and mechanism used to exchange information between levels. Consequently, hierarchical attention cannot be characterized by a single complexity expression that is independent of its architectural configuration [12].

An additional consideration is global dependency capture. Restricting high-resolution interactions can reduce computational requirements, but the architecture must provide an effective pathway through which information can propagate across distant regions. In hierarchical architectures, this propagation may occur through shifted windows, successive stages, downsampling, or cross-level representations. The computational benefit must therefore be considered jointly with the connectivity structure through which long-range dependencies are represented [12].

4.8. Comparative Computational Characteristics

A more fine-grained comparison of three important families of approximate attention, namely low-rank, kernel-based, and landmark-based mechanisms is given in Table 4 as follows:

Table 4. Comparative Characteristics of Approximate and Structured Self-Attention Mechanisms.

Method	Attention Class	Main Mechanism	Control Parameter	Dominant Computational Structure	Full n Times n Attention Matrix	Approximation or Structural Assumption	Global Interaction
Linformer [7]	Low-rank projection	Projects attention representations into a lower-dimensional space	Reduced rank r	Reduced-dimensional attention computation	No	Attention representations can be effectively represented in a lower-dimensional subspace	Global interaction through projected representation
Performer [10]	Kernel/feature-based	Replaces explicit softmax attention with a kernel feature representation	Feature dimension p	Associative feature-map computation	No	Feature representation approximates the softmax attention kernel	Global interaction through the kernel formulation
Nyströmformer [8]	Landmark/Nyström	Approximates attention using a selected set of landmark positions	Number of landmarks m	Landmark-based approximation and reconstruction	No	Attention structure can be approximated from representative landmark positions	Approximated global interaction
Reformer [13]	Hash-based structured attention	Groups likely interacting tokens using locality-sensitive hashing	Number/configuration of hash buckets	Hash-based restricted attention	No	Similar or relevant tokens are likely to be assigned to common buckets	Restricted direct interaction

The distinction among these mechanisms is necessary because they may all reduce the computational burden associated with dense self-attention, but they do so through different computational constructions and assumptions [7,8,10,13]. Their efficiency mechanisms and sources of approximation or restricted connectivity are therefore not equivalent. In particular, Linformer represents attention through reduced-dimensional projections, Performer reformulates the attention computation using kernel feature representations, whereas Nyströmformer approximates the attention structure using a selected set of landmark positions. Reformer differs from these approaches by using locality-sensitive hashing to restrict attention to groups of likely interacting tokens [7,8,10,13]. Treating these mechanisms as a single low-rank category would therefore obscure meaningful differences in their computational construction, parameter dependence, approximation assumptions, and error or connectivity characteristics. The approximation mechanisms considered in this review introduce different sources of representational error. Projection-based low-rank methods constrain the representation through a reduced-dimensional subspace; their behaviour therefore depends on the chosen rank and the information preserved by the projection. Landmark-based methods approximate the attention structure through a selected set of representative positions, making landmark count and selection strategy important determinants of approximation quality. Kernel or feature-based methods depend on the feature representation and on how closely the reformulated computation reproduces the intended attention function. Sparse and local methods reduce computation by restricting explicit interactions, which may limit direct information exchange between distant positions unless the architecture provides additional pathways. These mechanisms are therefore not interchangeable: each obtains efficiency through a different structural or numerical assumption, and its suitability depends on the task, sequence structure, and acceptable approximation or connectivity loss.

The comparison consequently reports the reduction parameter, dominant computational structure, whether a complete $n \times n$ attention matrix is materialized, and the principal

source of approximation or interaction restriction. This separation enables the mechanisms to be compared on common computational dimensions while preserving the theoretical distinctions that determine how and why each approach reduces the computational burden of dense attention.

5. Comparative Complexity Analysis

The taxonomy in Section 4 identifies the principal mechanisms through which self-attention computation can be reduced or reorganized while this section compares these mechanisms from a common computational perspective. Rather than revisiting their individual architectures, the analysis focuses on four dimensions that determine their complexity characteristics: sequence-length dependence, attention-related memory, approximation or exactness, and algorithm-specific control parameters. The comparison is formulated under the same dimensional assumptions introduced in Section 3, thereby avoiding the ambiguity that can arise when methods are described as "linear" without specifying which variables are treated as fixed.

5.1. Sequence-Length Dependence

For an input sequence of length n and representation dimension d, the conventional dense self-attention computation has the leading-order complexity

$$C_{\text{dense}} = O\left(nd^2 + n^2d\right) \tag{8}$$

The first term represents the dense query, key, and value projections, while the second represents the pairwise attention computation. When d is treated as fixed and n becomes sufficiently large, the n^2d component dominates. Thus, the characteristic scalability limitation of conventional self-attention arises from the number of token-to-token interactions, which grows quadratically with n [1,2].

The alternative mechanisms considered in this review modify this dependence in different ways. Under the assumption that their respective algorithm-specific parameters remain independent of n, representative complexity forms can be expressed as follows. For window- or pattern-restricted attention, a fixed local window of size w gives

$$C_{\text{sparse}} = O(nwd). \tag{9}$$

For reduced-rank projection-based attention, a representative reduced rank r gives

$$C_{\text{low-rank}} = O(nrd) \tag{10}$$

For representative landmark-based formulations, the computational behaviour depends on the number of landmarks m and the specific approximation and reconstruction procedure. For this reason, the landmark-based case is reported separately rather than being assigned the generic $O(nrd)$ form used for projection-based low-rank attention, since m and the associated landmark approximation operations determine its computational structure [8]. Here w,r,p, and m denote the local window size, reduced rank, feature-map dimension, and number of landmarks, respectively.

These expressions should not be interpreted as interchangeable definitions of linear attention. They describe different computational mechanisms. In particular w,r,p, and m correspond to different forms of restriction or reduction. A method is linear in sequence length only under the explicit assumption that its additional parameters do not increase with n. If, for example w,r, or p increases with sequence length, the effective scaling changes

accordingly. For landmark-based methods, the scaling also depends on m and the specific landmark approximation procedure [8].

This observation is important when comparing theoretical complexity. Two methods may both be described as linear with respect to sequence length while having substantially different computational costs because their constants, auxiliary dimensions, approximation procedures, and additional operations differ [2,3].

5.2. Attention-Memory Dependence

Time complexity alone does not provide a complete description of the computational burden of self-attention. The storage associated with attention interactions is particularly important because dense attention can require an explicit $n \times n$ attention structure. Its attention-matrix storage is therefore

$$M_{\text{dense}} = O\left(n^2\right). \tag{11}$$

Mechanisms that restrict attention to w interactions per token can reduce the corresponding interaction storage to approximately

$$M_{\text{sparse}} = O(nw), \tag{12}$$

provided that the implementation stores only the selected interactions rather than materializing a dense $n \times n$ structure [4–6].

Similarly, reduced representations and landmark formulations can avoid storing a complete attention matrix by operating through lower-dimensional representations. Their memory requirements, however, depend on the particular factorization, projection, reconstruction, and intermediate tensors used by the implementation [7,8]. This distinction is important because a sparse mathematical attention pattern does not necessarily imply sparse physical memory usage. An implementation that constructs a dense mask or dense intermediate tensor can retain quadratic memory behaviour even when the underlying attention pattern is sparse [2,3]. Consequently, memory complexity should be reported with reference to the actual computational representation rather than inferred solely from the conceptual attention pattern. Memory-efficient exact implementations represent a different case. Their objective is to avoid unnecessary materialization of intermediate attention structures while retaining the exact attention computation. Their improvement therefore arises primarily from the organization of memory access, data movement, and intermediate storage rather than from changing the asymptotic number of pairwise attention operations [3,11].

5.3. Exactness and Approximation

The computational reductions identified in Section 4 do not all have the same theoretical status. Some mechanisms reduce the number of interactions while preserving the attention operation over the selected pattern, whereas others approximate the full attention computation. The preceding mechanism-level comparison is extended by examining the computational and representational consequences of the different attention strategies in Table 5 as follows:

Table 5. Exact and Approximate Attention Mechanisms: Computational and Representational Characteristics.

Mechanism	Exactness	Full Attention Matrix Materialized?	Global Token Interaction	Main Source of Efficiency	Main Theoretical Tradeoff
Dense self-attention [1]	Exact	Yes, conceptually; not necessarily in every implementation	Yes	None	Quadratic interaction cost
Local/window attention [4]	Exact within prescribed pattern	No full matrix required	Restricted	Interaction restriction	Limited direct long-range interaction
Longformer [5]	Exact within defined sparse pattern	No	Partial global connectivity	Structured sparsity	Restricted interaction pattern
BigBird [6]	Exact within defined sparse pattern	No	Sparse global connectivity	Structured sparsity	Restricted interaction pattern
Linformer [7]	Approximate	No	Global interaction through projected representation	Low-rank projection	Approximation error associated with projection
Performer [10]	Approximate	No	Global interaction through kernel formulation	Feature-map reformulation	Kernel/feature approximation error
Nyströmformer [8]	Approximate	No	Approximate global interaction	Landmark representation	Landmark selection and reconstruction error
FlashAttention [3]	Exact	No full matrix materialization required	Yes	Tiling and I/O-aware execution	Does not reduce quadratic arithmetic complexity
FlashAttention-2 [11]	Exact	No full matrix materialization required	Yes	Improved parallelism and work partitioning	Does not fundamentally alter attention arithmetic
FlashAttention-3 [19]	Exact attention algorithm; reduced-precision execution may introduce numerical error	No	Yes	Asynchronous execution, warp specialization, and overlapping computation with data movement	Retains quadratic attention arithmetic; performance gains depend on GPU hardware, precision, and implementation
Swin Transformer [12]	Exact within defined windows	No global matrix	Hierarchical/ local	Windowed/ shifted-window attention	Restricted direct interaction

The distinction between exact and approximate computation is particularly important because a reduction in computational cost can arise from changing the interaction pattern, approximating the attention function, or reorganizing the execution of an otherwise exact computation [2,3]. The table therefore considers not only whether a method is exact or approximate, but also whether a complete attention matrix is materialized, whether unrestricted global interaction is retained, and what principal trade-off accompanies the efficiency gain.

Sparse and window-based mechanisms reduce computational requirements primarily by restricting the set of directly interacting positions [4–6], whereas Linformer, Performer, and Nyströmformer reduce computational requirements through different forms of approximation [7,8,10]. In contrast, FlashAttention and FlashAttention-2 preserve exact attention computation while reducing unnecessary intermediate memory materialisation and improving data movement and parallel execution [3,11]. FlashAttention-3 further advances this approach through asynchronous GPU execution, warp specialisation, and overlapping computation with data movement, while retaining quadratic arithmetic complexity for full attention [19]. Swin Transformer similarly preserves exact attention within its prescribed window structure, but its hierarchical and shifted-window design restricts direct interactions between distant regions [12].

By presenting these properties together, Table 5 provides a common basis for interpreting the different sources of efficiency. It also prevents mechanisms with fundamentally different computational principles from being treated as equivalent merely because they exhibit a lower asymptotic or practical resource requirement.

5.4. Effect of Algorithm-Specific Parameters

The theoretical advantage of efficient attention mechanisms depends strongly on the parameters that control the degree of computational reduction. For sparse attention, the principal parameter is the number of interactions per token, denoted by w. For reduced-rank approaches, the corresponding parameter is the reduced rank r, while landmark-based approaches depend on the number of landmarks m.

For example, comparing the dominant pairwise components of dense attention and a representative reduced-rank formulation gives

$$C_{\text{dense,pairwise}} = O\left(n^2 d\right) \tag{13}$$

and

$$C_{\text{low-rank}} = O(nrd). \tag{14}$$

Ignoring other implementation-specific terms, the reduced-rank formulation has a lower pairwise computational cost when

$$nrd < n^2 d. \tag{15}$$

For positive n,r, and d, this reduces to

$$r < n. \tag{16}$$

The same principle applies to sparse attention. Under the simplified comparison,

$$C_{\text{sparse,pairwise}} = O(nwd), \tag{17}$$

the interaction computation is reduced relative to dense attention when

$$nwd < n^2 d. \tag{18}$$

For positive n,w, and d, this gives

$$w < n. \tag{19}$$

Again, this condition describes an asymptotic relationship rather than a measured runtime advantage. A method with a smaller theoretical interaction count may still incur

additional overhead associated with indexing, irregular memory access, kernel launches, or other implementation operations [3,11].

5.5. Crossover Conditions and the Limits of Big-O Comparison

The preceding analysis demonstrates why asymptotic notation should not be used as the sole criterion for selecting an attention mechanism. Suppose a dense formulation has an effective computational model

$$C_{\text{dense}} = and^2 + bn^2d \tag{20}$$

while an alternative reduced-complexity mechanism has

$$C_{\text{eff}} = cnd^2 + enrd + f. \tag{21}$$

Here, a,b,c, and e are implementation-dependent constants, r is the reduction parameter, and f represents additional method-specific operations that are not captured by the dominant sequence-length terms. The dense formulation therefore contains both projection and pairwise attention costs, whereas the alternative formulation replaces the quadratic pairwise component with a reduced interaction term.

The theoretical asymptotic orders alone do not determine the crossover sequence length n value at which $C_{\text{eff}} < C_{\text{dense}}$. Ignoring the additional term f for a simplified analytical comparison, the crossover condition can be expressed as

$$and^2 + bn^2d > cnd^2 + enrd, \tag{22}$$

which, for $n > 0$ and $d > 0$, gives

$$n > \frac{(c-a)d + er}{b} \tag{23}$$

Equation (23) shows that n depends not only on the reduction parameter r, but also on the representation dimension d and the relative computational constants of the two implementations. When f is non-negligible, the crossover point can shift further because additional projection, reconstruction, normalization, feature-transformation, indexing, or memory-management operations increase the actual cost of the alternative mechanism.

This provides a formal explanation for an important practical observation: a method with a lower asymptotic dependence on sequence length may not outperform dense attention for short or moderate sequences. The additional operations required to construct reduced representations, select landmarks, transform features, manage sparse patterns, or coordinate memory accesses may offset the reduction in pairwise computation [2,3,11].

Accordingly, the relevant question is not simply whether the alternative has a lower asymptotic order, but under what combinations of n,d, the reduction parameter, and implementation overhead its actual computational cost becomes lower than that of dense attention. For sparse mechanisms, the corresponding reduction parameter is the local or effective interaction width w; for kernel-based approaches, it is the feature dimension p; and for landmark-based methods, it is the number of landmarks m. These parameters should therefore be interpreted according to the specific computational mechanism rather than treated as interchangeable forms of reduction [7–10].

The resulting crossover behaviour is mechanism-dependent. For low-rank projection methods, the dominant pairwise reduction is governed by the relationship between sequence length w and reduced rank r. For local or sparse attention, the corresponding relationship is between n and the effective interaction width w. Kernel- or feature-based

approaches introduce the feature dimension p, while landmark-based approaches depend on the number of landmarks m and on the additional operations required for landmark construction and attention reconstruction. Consequently, an efficient mechanism should not be considered practically advantageous merely because its asymptotic expression has a lower order in n. The relative values of n, d, and the mechanism-specific reduction parameter, together with implementation overhead, determine the computational regime in which the theoretical reduction becomes meaningful.

This distinction is particularly important when comparing algorithms across different sequence-length regimes. An efficient mechanism that is advantageous at very long sequences may provide little practical benefit at moderate sequence lengths, while an implementation optimized for memory traffic may provide benefits even when its arithmetic complexity remains quadratic [3,11].

For a finite-length comparison, let the dense-attention cost be represented by $C_D(n, d) = an^2d + bnd^2$, and let the reduced-complexity method have cost $C_R(n, d, r, s) = cnsd + end^2 + H(n, d, r, s)$, where r and s denote method-specific reduction parameters and H represents additional operations not captured by the leading interaction term. The coefficients describe operation-count conventions rather than measured hardware constants. A theoretical crossover occurs when: cost $C_R(n, d, r, s) < C_D(n, d)$. The resulting condition depends on the selected algorithm, the values and scaling of its control parameters, and the overhead terms included in the model. It identifies a crossover in the modelled operation count, not a guaranteed crossover in wall-clock execution time.

To compare the analytical operation counts on a common basis, the relative cost of each reduced-complexity method is defined as its modelled cost divided by the corresponding dense-attention cost. A ratio below one indicates a lower operation count under the assumptions of the model, whereas a ratio above one indicates a higher operation count. This ratio is an analytical comparison and should not be interpreted directly as a measured runtime ratio.

$$\frac{n^2d}{nrd} = \frac{n}{r}, \tag{24}$$

$$\frac{n^2d}{nwd} = \frac{n}{w}, \tag{25}$$

These ratios illustrate why the advantage of reduction becomes more pronounced as n increases. However, they do not represent total execution time because projection costs, auxiliary operations, memory traffic, implementation constants, and hardware utilization remain outside these simplified ratios [2,3,11].

The theoretical crossover discussed above can be illustrated by considering representative sequence lengths, representation dimensions, and reduction ranks. Table 6 compares the dominant pairwise interaction term of dense self-attention, n^2d, with the corresponding low-rank interaction term, nrd. The examples show how the relative arithmetic burden changes as the sequence length increases and as the reduction rank varies. The resulting ratios are intended only to illustrate the theoretical scaling relationship and should not be interpreted as measured execution-time improvements, since projection costs, implementation constants, memory traffic, and hardware-dependent overheads are excluded. The same analytical reasoning applies to sparse attention, for which the reduction parameter is the interaction or window size w with the dominant pairwise term represented by nwd rather than nrd.

Table 6. Illustrative Comparison of Dense and Low-Rank Pairwise Computational Costs.

Scenario	Sequence Length n	Representation Dimension d	Reduction Rank r	Dense Interaction n^2d	Low-Rank Interaction nrd	Dense /Low-Rank Ratio
Small sequence	512 *	64	64	16,777,216	2,097,152	8.00
Moderate sequence	2048	64	128	268,435,456	16,777,216	16.00
Large sequence	8192	64	256	4,294,967,296	134,217,728	32.00

* *The values in the table represent the dominant pairwise interaction terms only and are analytical illustrations rather than empirical measurements. The ratio is calculated as* $n^2d/(nrd) = n/r$.

5.6. Global Connectivity and Interaction Structure

Computational reduction can also alter the connectivity structure through which information is exchanged. Dense self-attention permits every token to interact directly with every other token, providing unrestricted pairwise connectivity [1]. Sparse and windowed mechanisms deliberately restrict this interaction structure by limiting the set of token pairs that participate in attention [4–6]. Although this reduces the number of pairwise computations, it can also limit direct long-range interactions unless additional connectivity mechanisms are introduced [4–6].

Landmark and hierarchical mechanisms address this issue differently by introducing representative positions or multiple levels of interaction. Landmark-based approaches approximate global attention through a smaller set of representative positions [8], whereas hierarchical approaches organize interactions across local regions and multiple representational levels [12]. Kernel and linear formulations instead reorganize the attention computation so that global aggregation can be obtained without explicitly constructing the full pairwise attention matrix [9,10].

This demonstrates that computational efficiency and connectivity are related but distinct dimensions. Reducing the number of interactions can lower computational cost, but the resulting information-propagation structure must also be considered when evaluating the consequences of the reduction. In particular, lower interaction density does not necessarily imply equivalent access to long-range dependencies, making connectivity an important dimension for comparing efficient attention mechanisms.

5.7. Hardware and Implementation Dependence

The preceding mathematical expressions describe algorithmic complexity, but practical execution also depends on the computational environment. Memory bandwidth, cache behaviour, accelerator architecture, parallelization, kernel fusion, and data movement can influence execution time independently of asymptotic operation counts [3,11].

This distinction is particularly evident for memory-efficient exact attention. An implementation can retain the quadratic arithmetic structure of dense attention while reducing the amount of intermediate data transferred between memory levels. Consequently, its practical execution characteristics cannot be inferred solely from the arithmetic complexity of the attention operation [3,11].

Conversely, a theoretically linear method may require additional transformations, feature mappings, or irregular memory accesses that reduce the practical benefit predicted by its asymptotic expression [2,3]. Thus, the computational hierarchy relevant to efficient attention extends beyond arithmetic operation count to include memory traffic, intermediate storage, parallel execution, and hardware utilization [3,11].

The comparison indicates that no single efficiency strategy is uniformly preferable. Sparse and local mechanisms are attractive when the interaction pattern can be restricted without undermining the task's need for long-range information exchange. Projection-based and landmark-based approximations are useful when a reduced representation can preserve the information needed for the target computation, but their benefits depend on the reduction parameter and approximation procedure. Kernel or linear attention can reduce sequence-dependent computation under the assumptions of the chosen formulation, although the feature representation and any approximation to standard softmax attention remain relevant. Memory-efficient exact methods are particularly important when the principal bottleneck is intermediate storage or data movement rather than the number of pairwise arithmetic operations. These distinctions imply that algorithm selection should begin by identifying the dominant bottleneck and the acceptable changes to fidelity or connectivity, rather than by selecting a method solely because its asymptotic expression has a lower order.

6. Practical Efficiency and Hardware Considerations

The complexity expressions developed in the preceding sections characterize the asymptotic growth of practical memory and execution requirements, but they do not fully determine the practical efficiency of an attention algorithm. Actual execution depends on factors that are not represented explicitly by Big-O notation, including constant factors, memory bandwidth, data movement, parallelization, kernel implementation, accelerator architecture, and intermediate storage. The practical memory and execution requirements of attention also depend on the computational regime. During training, the memory footprint can include intermediate tensors and activations retained for gradient computation, whereas inference may involve prompt processing and repeated autoregressive decoding. These regimes differ in their sequence-length patterns, reuse of previously computed representations, and memory requirements. Consequently, attention-intermediate storage should not be equated with total training memory or the complete memory footprint of an inference system. The present comparison focuses on attention-related computational and memory behaviour; broader system-level costs depend on the model, runtime, and deployment configuration.

6.1. Arithmetic Complexity Versus Execution Cost

Asymptotic complexity provides a useful description of how computational requirements grow as sequence length increases. For dense self-attention, the dominant pairwise component grows according to

$$O\left(n^2 d\right),$$

whereas representative sparse, reduced-rank, and feature-based formulations can exhibit dependencies such as

$$O(nwd),$$

and

$$O(nrd),$$

respectively. Kernel- or feature-map-based formulations may instead be expressed in terms of the feature dimension p, giving a representative form of $O(npd)$. These expressions establish different growth rates under their respective assumptions. They do not, however, specify the number of processor cycles, memory transfers, kernel launches, or synchronization operations required by an implementation [2,3,11].

A simplified execution model can be expressed as

$$T_{\text{exec}} = T_{\text{arith}} + T_{\text{memory}} + T_{\text{overhead}}, \tag{26}$$

where T_{arith} represents the time associated with arithmetic operations, T_{memory} represents data movement and memory-access costs, and T_{overhead} represents implementation-specific costs such as kernel launches, indexing, synchronization, and auxiliary transformations [3,11].

This formulation illustrates why algorithms with different asymptotic complexities cannot be assumed to have the same practical performance ordering across all sequence lengths. An algorithm with fewer arithmetic operations may incur greater memory or implementation overhead, whereas an algorithm with a higher arithmetic count may benefit from highly optimized dense matrix operations [3,11].

6.2. Memory Bandwidth and Data Movement

Memory behaviour becomes particularly important for attention because intermediate tensors can be substantially larger than the input representation. Dense attention can generate an $n \times n$ attention structure, creating both storage requirements and data-movement requirements that increase quadratically with sequence length [2,3].

Reducing the amount of intermediate data can therefore provide practical benefits even when the arithmetic complexity remains unchanged. Memory-efficient exact attention methods exploit this distinction by reorganizing computation so that intermediate attention information does not need to be fully materialized in high-capacity memory [3,11].

The practical significance of such an approach depends on the relative cost of arithmetic and memory operations on the target hardware. When execution is constrained primarily by memory traffic rather than arithmetic throughput, reducing data movement can have a substantial effect on performance even without changing the asymptotic operation count [3,11].

This observation also explains why theoretical memory complexity and physical memory traffic should not be treated as identical quantities. An algorithm may require only a limited amount of simultaneously resident intermediate data while still performing substantial data movement during its execution [3,11].

6.3. Dense Operations and Hardware Utilization

Modern accelerators are highly optimized for dense matrix operations. Consequently, an algorithm with a theoretically lower number of arithmetic operations does not necessarily achieve a proportional reduction in execution time if its computation is irregular or poorly aligned with the underlying hardware [3,11].

Dense self-attention benefits from highly optimized matrix multiplication routines and parallel execution. Sparse mechanisms can reduce the number of arithmetic operations, but irregular sparsity patterns may introduce indexing and memory-access overhead. The practical benefit of sparsity therefore depends partly on whether the hardware and implementation can exploit the reduced interaction structure efficiently [2,3].

Structured sparsity can provide a more predictable computational pattern than arbitrary sparsity. Local windows, blocks, and other regular structures can facilitate efficient memory access and parallel execution. The distinction between theoretical sparsity and hardware-exploitable sparsity is therefore important when interpreting reported efficiency improvements [2,3].

6.4. Exact Memory-Efficient Attention

Memory-efficient exact attention illustrates the difference between algorithmic complexity and implementation efficiency particularly clearly. An exact implementation may retain the quadratic pairwise computation $O(n^2d)$ while avoiding explicit materialization of the complete $n \times n$ attention matrix [3,11]. The practical improvement is obtained through the organization of computation, including tiling and more efficient movement of intermediate data [3,11]. Consequently, such methods should not be placed in the same complexity category as approaches that reduce the number of pairwise interactions or approximate the attention function.

This distinction is important when comparing exact memory-efficient attention with sparse, low-rank, landmark, or kernel-based mechanisms. The former primarily changes how the computation is executed and how intermediate data are handled, whereas the latter mechanisms generally change the computational structure or representation used to obtain attention [2,3,7–10].

The distinction also prevents an inappropriate comparison in which a hardware-aware implementation is described as having lower asymptotic attention complexity merely because it has lower intermediate memory requirements. Reduced memory consumption and reduced arithmetic complexity are related but distinct properties [2,3,11].

6.5. Practical Implications of Sparse Attention

Sparse attention illustrates another limitation of asymptotic analysis. Consider a local attention mechanism with window size w, giving a dominant pairwise interaction complexity of $O(nwd)$. Relative to dense self-attention, whose corresponding pairwise component is $O(n^2d)$, the theoretical reduction becomes larger as the sequence length n increases relative to the window size w [2,4–6].

The practical benefit, however, depends on whether the implementation can exploit the sparse structure efficiently. For highly regular local windows, the computational pattern is relatively predictable and can be implemented using structured memory access and parallel execution. In contrast, irregular sparse patterns may require additional indexing, gather, scatter, or routing operations. These operations can increase memory traffic and reduce parallel efficiency [2,3].

Furthermore, restricting attention changes the connectivity structure through which information is exchanged. A smaller computational footprint can therefore involve a representational trade-off that does not arise in the same form for memory-efficient implementations of dense attention. Sparse attention should consequently be evaluated not only in terms of its reduction in pairwise computation but also in terms of the connectivity it preserves and the implementation overhead required to exploit the sparse pattern [4–6].

6.6. Practical Implications of Approximate Attention

Low-rank, landmark, and kernel-based mechanisms introduce different forms of computational and representational trade-offs. Their computational advantages are obtained by replacing, approximating, or reorganizing aspects of the original attention calculation. The resulting computational cost depends on mechanism-specific parameters such as the reduced rank r, the number of landmarks m, or the feature dimension p [7–10].

For example, reducing the representation from the original dimension n to a substantially smaller rank r can reduce the dominant pairwise interaction cost from $O(n^2d)$ to a representative form of $O(nrd)$ for low-rank projection methods such as Linformer [7]. However, the computational savings must be considered together with the additional operations required to construct the reduced representation and, where applicable, reconstruct or approximate the final attention output. Landmark-based methods such as Nyströmformer

involve a different approximation procedure based on representative positions, so their computational cost depends on the landmark construction and reconstruction process rather than following the same expression as direct low-rank projection methods [8].

More importantly, approximation introduces an additional dimension that is absent from exact memory-optimized attention. A reduction in computational cost may therefore be accompanied by approximation error. The appropriate theoretical comparison is not simply lower computational cost versus higher computational cost, but rather a multidimensional relationship among computational cost, memory consumption, approximation error, and connectivity [7–10].

6.7. Moderate Sequence Lengths

The theoretical advantage of reduced-complexity attention becomes increasingly apparent as sequence length grows, but this does not imply that an alternative mechanism will always be practically preferable at moderate sequence lengths. For dense attention, the pairwise component is proportional to n^2d. For a reduced-rank mechanism, a representative pairwise component may be proportional to nrd. Their ratio is approximately $\frac{n^2d}{nrd} = \frac{n}{r}$ as can be seen in Equation (25). This ratio indicates that the theoretical arithmetic advantage grows with n/r. When n is only modestly larger than r, the reduction in the dominant pairwise computation may be relatively limited. Additional projection, approximation, indexing, memory, or implementation overhead can therefore become important [2,3,7,11].

The same reasoning applies to local attention, for which the corresponding ratio is approximately $\frac{n^2d}{nwd} = \frac{n}{w}$ as can be seen in Equation (26). These relationships provide a useful theoretical explanation for why sequence length alone is insufficient for selecting an efficient attention mechanism. The relevant computational regime is determined jointly by n,d and the mechanism-specific parameters.

6.8. Hardware Dependence as a Comparative Dimension

Hardware dependence should therefore be treated as an explicit dimension of attention efficiency rather than as an implementation detail that can be ignored in theoretical comparison. The same algorithm can exhibit different practical behaviour on different processors or accelerators because the relative costs of arithmetic operations, memory access, data movement, communication, and synchronization vary across hardware environments [3,11].

A computational mechanism that is theoretically attractive but poorly aligned with the available hardware may provide less practical benefit than expected. Conversely, an algorithm that retains a higher asymptotic arithmetic complexity may benefit from highly optimized dense operations, parallel execution, and efficient memory management [3,11]. This observation is particularly relevant when comparing architectural efficiency with kernel-level or hardware-aware optimization. Architectural mechanisms alter the mathematical computation or representation used to obtain attention, whereas implementation-level optimizations can improve the execution of a given mathematical computation without necessarily changing its asymptotic formulation [2,3,11]. Distinguishing these levels of efficiency is therefore important when interpreting theoretical complexity reductions and reported practical performance.

6.9. Implications for Algorithm Comparison

The analysis suggests that efficient self-attention should be evaluated across at least four interconnected dimensions: arithmetic cost, memory behaviour, approximation characteristics, and hardware execution [2,3,11].

These dimensions should not be collapsed into a single complexity label. A sparse method may have lower arithmetic complexity but restricted connectivity [4–6]. A low-rank

or landmark method may reduce computational cost at the expense of approximation [7,8]. A kernel-based method may avoid explicit construction of the complete attention matrix but introduce feature-transformation costs [9,10]. A memory-efficient exact method may retain quadratic arithmetic complexity while substantially improving memory behaviour and data movement [3,11]. Consequently, the computational efficiency of an attention mechanism is better understood as a multidimensional property than as a single asymptotic order. The most relevant comparison depends on sequence length, representation dimension, mechanism-specific reduction parameters, approximation requirements, connectivity requirements, and target hardware [2,3]. This perspective also establishes a basis for identifying unresolved theoretical problems. In particular, future analyses need to determine how computational savings, approximation error, memory traffic, and long-range dependency preservation interact across different combinations of sequence length and representation dimension. These issues are considered in the research questions developed in the subsequent section.

Table 7 summarizes the principal theoretical and practical factors that influence the efficiency of self-attention beyond asymptotic arithmetic complexity [2,3,7–11]. It highlights how memory behaviour, approximation, connectivity, implementation overhead, and hardware utilization can affect the practical efficiency of different attention mechanisms.

Table 7. Theoretical and Practical Factors Influencing Self-Attention Efficiency.

Efficiency Dimension	What It Measures	Dense Attention	Sparse /Structured Attention	Approximate Attention	Memory-Efficient Exact Attention
Arithmetic operations	Number of mathematical operations	High	Reduced	Reduced	Similar asymptotic arithmetic cost
Attention-memory requirement	Intermediate attention storage	High	Reduced *	Reduced *	Strongly reduced
Memory traffic	Data movement between memory levels	High	Pattern-dependent	Method-dependent	Reduced through tiling/reuse
Interaction coverage	Direct token-to-token connectivity	Global	Restricted/structured	Approximated global interaction	Global
Approximation error	Deviation from exact attention	None	None for defined sparse pattern	Present	None
Hardware dependence	Sensitivity to implementation/hardware	High	High	High	High
Implementation overhead	Indexing, projection, feature transformation, and related operations	Relatively low	Pattern-dependent	Often additional	Tiling/kernel-related
Main efficiency mechanism	Primary source of improvement	None	Interaction reduction	Mathematical approximation/reformulation	I/O and memory optimization

* *The memory reduction assumes that only the selected sparse interactions are stored and that the implementation does not materialize a dense* $n \times n$ *attention structure.*

7. Research Gaps and Open Questions

The unresolved issue is not simply whether efficient attention can reduce asymptotic complexity, but how the conditions and costs of that reduction can be characterised consistently. First, finite-length crossover models require operation-count terms that account for projection work and method-specific overhead rather than relying on asymptotic notation alone. Second, the relationship between reduction parameters and approximation quality remains formulation-dependent and requires evaluation with clearly defined fidelity or task metrics. Third, attention-related memory and I/O costs need to be separated from total model memory and from arithmetic operation counts. Fourth, restricted connectivity must be analysed in relation to the pathways available for preserving long-range information exchange. Finally, hardware-aware comparisons require reported workload, precision, accelerator, and implementation details so that analytical complexity and measured execution behaviour can be interpreted together.

7.1. Need for a Unified Multidimensional Evaluation Framework

Existing complexity analyses commonly emphasize arithmetic operation count and asymptotic sequence-length dependence. Although these measures are important, they do not fully capture the factors that determine the theoretical and practical behaviour of efficient attention [2,3,11].

A more comprehensive framework should jointly represent at least the following dimensions:

$$\mathcal{E} = \left\{ C_{\text{time}}, C_{\text{memory}}, E_{\text{approx}}, B_{\text{I/O}}, G_{\text{connectivity}}, H_{\text{hardware}} \right\}, \tag{27}$$

where C_{time} denotes computational complexity, C_{memory} denotes memory requirements, E_{approx} represents approximation error, $B_{I/O}$ represents memory-bandwidth and data-movement requirements, $G_{\text{connectivity}}$ represents the resulting interaction structure, and H_{hardware} represents hardware dependence.

A key research question is therefore:

RQ1: How can a unified theoretical framework simultaneously characterize arithmetic complexity, memory consumption, approximation behaviour, I/O requirements, connectivity, and hardware dependence across different self-attention mechanisms?

Such a framework would make it possible to compare methods that achieve efficiency through fundamentally different mechanisms without reducing all of them to a single Big-O expression [2,3,7–11].

7.2. Crossover Conditions Between Dense and Efficient Attention

The analysis in Sections 5 and 6 shows that asymptotic superiority does not necessarily imply practical superiority at every sequence length. The crossover between two mechanisms depends on sequence length, representation dimension, reduction parameters, implementation constants, and additional computational overhead [2,3,7–11].

This leads to the following question:

RQ2: Under what combinations of sequence length n, representation dimension d, and algorithm-specific parameters such as reduced rank r, window size w, or number of landmarks m does an efficient attention mechanism become computationally advantageous over dense self-attention?

A useful theoretical treatment should derive crossover conditions rather than simply state that one method is linear and another is quadratic. Such an analysis should also distinguish the idealized asymptotic crossover from the practical crossover obtained when projection, indexing, feature transformation, reconstruction, and memory-access costs are included [2,3,7–11].

7.3. Approximation Error Versus Computational Reduction

Approximation-based mechanisms introduce a fundamental trade-off between computational reduction and fidelity to the original attention operation. Low-rank, landmark-based, and kernel-based approaches can reduce or reorganize the computational burden of attention, but the relationship between their reduction parameters and approximation error is not uniform [7–10].

This motivates:

RQ3: How does the reduction in computational and memory requirements relate quantitatively to approximation error across low-rank, landmark-based, and kernel-based attention mechanisms?

In particular, future work should investigate whether approximation error can be characterized using theoretically comparable measures across different approximation mechanisms. Such analysis could establish whether two methods with similar computational requirements provide comparable preservation of the original attention function. It should also examine how the relevant reduction parameters, such as reduced rank r, number of landmarks m, and feature dimension p, influence both computational requirements and approximation error [7–10].

7.4. Memory Bandwidth and I/O Complexity

Arithmetic operation count does not fully characterize computational cost on modern accelerators. Memory movement can become a major determinant of execution time, particularly when large intermediate attention structures are repeatedly written to and retrieved from memory [3,11].

A corresponding research question is:

RQ4: How should memory bandwidth, data movement, and intermediate-tensor traffic be incorporated into the theoretical complexity analysis of self-attention?

This question is especially relevant when comparing architectural approaches that reduce the number of arithmetic operations with implementation-level approaches that preserve the mathematical computation but reduce memory traffic [3,11]. A useful framework should therefore distinguish arithmetic complexity from I/O complexity rather than treating memory only as a storage-capacity constraint.

7.5. Long-Range Dependency and Connectivity

Restricting attention can reduce computational cost but may also alter the connectivity through which information propagates across a sequence. Local and sparse mechanisms therefore introduce a question that is not captured by conventional complexity notation alone [4–6].

The relevant research question is:

RQ5: How can the ability of an efficient attention mechanism to preserve long-range dependencies be characterized alongside its computational and memory complexity?

A theoretical measure of connectivity could consider the effective interaction distance, the number of computational steps required for information to propagate between distant tokens, and the availability of direct or indirect global pathways. Such a measure should distinguish direct token-to-token interaction from information that can propagate only through intermediate tokens, landmarks, or hierarchical levels [5,6,8,12].

This would provide a more rigorous basis for comparing mechanisms that reduce computation by restricting interactions with mechanisms that retain global interaction through algebraic reformulation or hierarchical structures [9,10,12].

7.6. Pareto Trade-Offs Between Efficiency and Fidelity

The preceding questions indicate that attention mechanisms should not necessarily be compared using a single efficiency criterion. A mechanism can simultaneously have lower computational complexity, higher approximation error, lower memory consumption, or weaker direct connectivity.

This suggests a multi-objective formulation in which computational cost, memory consumption, approximation error, and connectivity are considered jointly. A simplified representation can be expressed as

$$\min \mathcal{F} = \left(C_{\text{time}} , C_{\text{memory}} , E_{\text{approx}} , -D_{\text{connectivity}} \right) \tag{28}$$

Here, C_{time}, C_{memory}, and E_{approx} are minimized, whereas $D_{\text{connectivity}}$ is maximized. The negative sign converts the connectivity objective to the same minimization form. It indicates that higher connectivity is desirable, whereas the other three quantities are minimized.

A method may therefore be considered Pareto-efficient when improving one objective necessarily involves deterioration in at least one other objective.

The corresponding research question is:

RQ6: What Pareto trade-offs exist among computational cost, memory consumption, approximation error, and long-range connectivity across efficient attention mechanisms?

Such an analysis could provide a more informative basis for comparing algorithms than ranking them according to a single asymptotic complexity class.

7.7. Dependence on Sequence and Representation Dimensions

The relative computational and memory costs of attention mechanisms depend jointly on sequence length n and representation dimension d. However, complexity discussions frequently emphasize n while treating d as an implicit constant.

This motivates:

RQ7: How do the relative computational and memory advantages of efficient attention mechanisms change across different combinations of sequence length n and representation dimension d?

This question is particularly important because an approach that reduces the dependence on n may introduce additional dependence on a reduced representation dimension, feature dimension, or projection dimension. Consequently, evaluating only the asymptotic dependence on n can obscure important regimes in which the representation dimension contributes substantially to the total computational cost.

7.8. Toward Hardware-Aware Complexity Models

Finally, the analysis indicates that algorithmic complexity and hardware execution should not be treated as entirely independent. Dense matrix operations, structured sparsity, memory-efficient tiling, and irregular interaction patterns can exhibit substantially different execution characteristics across hardware platforms [3,11].

This motivates a final research question:

RQ8: How can theoretical complexity models incorporate hardware-specific factors sufficiently to predict practical efficiency without becoming tied to a single hardware platform?

A useful model should retain the generality of asymptotic analysis while incorporating measurable factors such as memory traffic, parallelism, memory bandwidth, and accelerator utilization. Such a model could provide a more meaningful connection between theoretical computational complexity and practical execution efficiency while remaining sufficiently general to support comparisons across different hardware platforms [3,11].

7.9. Summary of Research Gaps

The identified gaps can be summarized as a progression from single-dimensional complexity analysis toward multidimensional computational characterization as shown in Table 8:

Table 8. Research Gaps and Open Questions in Computationally Efficient Self-Attention.

Research Gap	Specific Unresolved Issue	Key Variables/Measures	Proposed Research Question
Unified evaluation	Lack of a multidimensional complexity framework	Time, memory, I/O, approximation, connectivity	RQ1: How can these dimensions be jointly characterized?
Crossover analysis	Unclear conditions under which reduced complexity becomes practically advantageous	n, d, r, w, m, implementation overhead	RQ2: Under what parameter combinations does an efficient method become computationally advantageous over dense attention?
Approximation analysis	Computational savings versus approximation error	Rank, landmarks, feature dimension, approximation error	RQ3: How does computational reduction relate to approximation error?
I/O complexity	Arithmetic complexity does not fully capture memory movement	Bandwidth, data traffic, intermediate storage	RQ4: How should I/O cost be incorporated into complexity analysis?
Connectivity	Reduced interaction may affect long-range information propagation	Interaction distance, connectivity, propagation paths	RQ5: How can long-range dependency preservation be quantified?
Pareto efficiency	Multiple competing efficiency objectives	Time, memory, error, connectivity	RQ6: What trade-offs define Pareto-efficient attention mechanisms?
$n - d$ dependence	Sequence length is often analyzed independently of representation dimension	Sequence length n, representation dimension d, method-specific dimensions	RQ7: How do efficiency relationships change jointly with n and d?
Hardware-aware complexity	Asymptotic models do not fully predict practical execution	Bandwidth, parallelism, memory traffic, accelerator utilization	RQ8: How can hardware effects be incorporated into general complexity models?

The identified gaps indicate that the central unresolved issue is not simply how to reduce the asymptotic complexity of self-attention. Rather, it is how to characterize and compare computational efficiency when arithmetic cost, memory behaviour, approximation, connectivity, and hardware execution interact. Addressing this problem would provide a stronger theoretical basis for evaluating existing and future efficient attention mechanisms.

8. Discussion

The analysis presented in this review indicates that efficient self-attention should be understood as a multidimensional computational problem rather than as a simple progression from quadratic to linear complexity. The reviewed mechanisms reduce computational burden through substantially different strategies, including restricting token interactions,

reducing the dimensionality of the attention representation, approximating the attention function, reorganizing computation, or reducing memory movement. These strategies affect different components of the computational process and therefore cannot be considered equivalent solely on the basis of their asymptotic dependence on sequence length.

A central observation is that the distinction between algorithmic complexity reduction and implementation-level optimization is essential. Sparse, low-rank, kernel-based, and landmark-based mechanisms modify the computation performed by the attention operation itself, whereas memory-efficient approaches such as FlashAttention primarily modify how the exact computation is executed [3,11]. The latter can substantially reduce memory requirements and improve practical execution without eliminating the quadratic pairwise arithmetic component. Consequently, classifying all efficient mechanisms according to a single complexity measure can conceal important theoretical and practical differences.

The analysis also shows that the frequently used distinction between quadratic and linear attention requires careful qualification. Expressions such as $O(nrd)$ or $O(npd)$ are linear in n only when the relevant reduction parameters are treated as fixed or sufficiently independent of sequence length. Similarly, the projection component $O(nd^2)$ remains present in many formulations even when the attention-interaction component is reduced. Thus, describing a mechanism simply as "linear attention" does not imply that its complete computational cost is linear in every model dimension or implementation setting [2,7,9,10].

The crossover analysis further indicates why asymptotic complexity should not be interpreted as a direct prediction of practical performance. For reduced-complexity mechanisms, the theoretical advantage depends on the relationship between sequence length and parameters such as reduced rank, feature dimension, landmark count, or local attention width. At moderate sequence lengths, additional projections, feature transformations, indexing operations, kernel launches, or memory-access costs may offset part of the theoretical reduction [2,3,7–11]. Practical comparison therefore requires consideration of both the growth rate of the dominant term and the fixed or lower-order costs associated with implementation.

Another important distinction concerns memory complexity and memory traffic. Reducing the amount of memory required to store intermediate attention information is not equivalent to reducing the number of arithmetic operations. Memory-efficient exact attention demonstrates that substantial efficiency gains can be obtained by avoiding explicit materialization of the full attention matrix and by reorganizing data movement [3,11]. This observation suggests that memory capacity, memory bandwidth, and arithmetic throughput should be treated as related but distinct dimensions in theoretical comparisons of attention mechanisms.

Approximation introduces an additional dimension to this analysis. Low-rank, kernel-based, and landmark-based methods obtain computational savings by replacing or approximating aspects of the original attention computation [7–10]. Their efficiency must therefore be considered together with the assumptions under which the approximation is valid and the resulting approximation error. A lower computational cost does not by itself establish that two mechanisms provide equivalent computational representations. Appropriate comparison should consider the relationship among computational reduction, approximation fidelity, and the structural properties of the resulting attention representation.

The treatment of sparse and hierarchical attention also highlights a representational trade-off. Restricting direct token interactions can reduce computational requirements substantially, but it changes the connectivity structure through which information is exchanged [4–6,12]. Local attention, structured sparse attention, and hierarchical mechanisms therefore require consideration of both interaction count and information propagation. An attention mechanism that reduces the number of direct interactions may still preserve

useful long-range information through indirect pathways, but the characteristics of those pathways differ from those of globally connected dense attention.

Taken together, these observations suggest that no single complexity class adequately characterizes the efficiency of all self-attention mechanisms. A more informative comparison should simultaneously consider computational cost, memory requirements, approximation behaviour, interaction structure, and execution characteristics [2,3,7–11]. This multidimensional perspective provides a more rigorous basis for interpreting claims of efficiency and avoids treating theoretical complexity reduction as synonymous with practical performance improvement.

Quantitative results from individual studies provide useful evidence of the trade-offs associated with approximate attention, although their interpretation depends on the evaluation setting. In the Long Range Arena benchmark, Xiong et al. [8] reported an average classification accuracy of 58.95% for Nyströmformer, compared with 58.77% for standard self-attention and 53.63% for Performer. These results were obtained using the authors' implementation of a two-layer Transformer with an embedding dimension of 64, a hidden dimension of 128, and two attention heads, across benchmark tasks with sequence lengths ranging from approximately 1000 to 4000 tokens. In a separate attention-module efficiency test on an NVIDIA GeForce GTX 1080 Ti, with an input sequence length of 8192, Nyströmformer using 32 landmarks required 383 MB of memory and 11.5 ms per input instance, compared with 10,233 MB and 155.4 ms for standard attention [8]. The results illustrate that approximation can affect both downstream performance and computational requirements, but they remain specific to the configurations and evaluation procedures used in the original study. They should not be interpreted as universal performance guarantees or as directly comparable with measurements from unrelated experiments.

A direct numerical comparison across the broader efficient-attention literature is not generally justified because related studies differ in task and dataset, sequence length, model architecture, approximation parameters, hardware, implementation, and evaluation metric. Approximation error, downstream accuracy, latency, and memory consumption also measure different properties and cannot be treated as interchangeable indicators of efficiency. Accordingly, the quantitative findings discussed here are used to illustrate results within their original experimental settings rather than to establish a universal ranking of attention mechanisms.

The principal contribution of this review is the integration of these otherwise distinct computational perspectives into a common analytical basis for comparing efficient self-attention mechanisms. Rather than proposing a new attention algorithm or empirical benchmark, the review clarifies how efficiency arises from different sources, including reducing arithmetic operations, restricting interactions, approximating attention, reorganizing memory movement, and exploiting hardware-aware execution. By relating these mechanisms to sequence length, representation dimension, method-specific parameters, memory behaviour, approximation, connectivity, and implementation conditions, the review provides a consistent basis for interpreting theoretical efficiency claims and identifying the conditions under which computational reductions may translate into practical advantage.

The quadratic pairwise cost of dense attention and the memory-aware execution principles of exact attention are established results in the reviewed literature; the additional contribution here is to interpret these results alongside the assumptions, control parameters, approximation characteristics, and overhead terms that determine the scope and limits of cross-method complexity comparisons. The review also reveals an important methodological implication for future comparative studies. Reported efficiency should ideally specify the sequence length, representation dimension, mechanism-specific parameters, exactness

or approximation status, memory requirements, and relevant implementation conditions. Without these details, comparisons across attention mechanisms may conflate fundamentally different sources of efficiency. A theoretical complexity result should therefore be interpreted within the assumptions under which it was derived and should not automatically be generalized to all sequence lengths, hardware platforms, or implementation environments.

Overall, the findings support a shift from single-metric complexity comparison toward multidimensional computational characterization. Such a perspective can help researchers distinguish whether an observed efficiency gain results from reducing computation, reducing data movement, approximating the underlying operation, restricting information exchange, or exploiting hardware more effectively. It also provides a clearer foundation for evaluating future attention mechanisms whose efficiency may arise from combinations of these strategies.

9. Limitations of the Study

This study has some limitations. First, it is a theoretical and comparative review rather than an experimental benchmarking study. The reported complexity expressions therefore describe analytical computational behaviour and should not be interpreted as direct measurements of wall-clock execution time, energy consumption, or hardware utilization [2,3,11].

Second, the practical behaviour of several mechanisms is implementation-dependent. Factors such as kernel design, accelerator architecture, memory hierarchy, parallelization strategy, and software optimization can affect observed performance without changing the underlying asymptotic complexity [3,11]. Consequently, the theoretical comparisons presented in this review provide a common analytical basis but cannot replace controlled empirical benchmarking.

Third, the complexity of some mechanisms depends on implementation-specific parameters and assumptions. For sparse, landmark-based, kernel-based, and hierarchical approaches, the effective cost can vary according to interaction patterns, reduction dimensions, landmark selection, feature representations, or window configurations [4–12]. The expressions reported in this review therefore represent the stated analytical assumptions rather than universal runtime equations applicable to every implementation.

Fourth, the crossover expressions and illustrative scenarios are analytical comparisons derived from stated cost assumptions; they are not calibrated estimates of wall-clock runtime or energy consumption. Actual performance can differ because of kernel implementation, hardware architecture, numerical precision, memory traffic, workload shape, and method-specific overhead. The review therefore uses these models to clarify the conditions under which computational advantages may arise, not to claim a universal measured break-even point for any attention algorithm.

Finally, the review focuses primarily on computational and memory characteristics. Although approximation error, connectivity, and hardware considerations are incorporated into the comparative discussion, they are not subjected to a common empirical measurement framework. Establishing such a framework would require systematic benchmarking across controlled combinations of sequence length, representation dimension, model architecture, and hardware platform.

These limitations do not diminish the analytical purpose of the review. Rather, they define the boundary within which the presented complexity comparisons should be interpreted and motivate the specific research questions identified in Section 7. They also suggest that the computational complexity of self-attention should not be regarded as a single problem with a single solution. Rather, it represents a family of related optimization

problems involving interaction count, representation dimensionality, memory usage, and computational execution.

10. Conclusions

This study establishes that efficient self-attention methods reduce computational burden through distinct mechanisms, including restricting token interactions, reducing representation dimensions, reformulating attention computation, and improving memory-aware execution. These mechanisms differ in their arithmetic costs, memory requirements, approximation assumptions, connectivity patterns, and dependence on implementation conditions.

The analytical interpretation developed in this review shows why these differences matter when comparing complexity claims. A lower asymptotic order does not, by itself, establish a lower operation count at every finite sequence length or guarantee faster execution on a particular accelerator. Meaningful comparisons must state the relevant dimensions, reduction parameters, included operations, and assumptions about implementation overhead.

Future work should develop and evaluate analytical and empirical frameworks that connect these explicit cost models with measured runtime, memory traffic, approximation quality, and hardware utilisation under reproducible workloads. Such frameworks would help determine when a theoretical reduction produces a practical advantage and what representational or execution trade-offs accompany it.

Author Contributions: Conceptualization, A.O.A.; literature search and selection, A.O.A.; theoretical analysis, A.O.A.; formal analysis, A.O.A.; investigation, A.O.A.; literature synthesis and comparative evaluation, A.O.A.; resources, A.O.A. and S.O.O.; writing—original draft preparation, A.O.A.; writing—review and editing, A.O.A. and S.O.O.; supervision, S.O.O.; project administration, A.O.A. and S.O.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: No new data were created or analyzed in this study. Data discussed in this review are available from the original publications cited in the manuscript.

Acknowledgments: During the preparation of this manuscript, the authors used GPT-6 for language refinement and organization of selected manuscript sections. The authors critically reviewed, and verified the assistance provided by named tool and take full responsibility for the accuracy, interpretation, and content of the published manuscript.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention Is All You Need. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 5998–6008.
2. Tay, Y.; Dehghani, M.; Bahri, D.; Metzler, D. Efficient Transformers: A Survey. *ACM Comput. Surv.* **2022**, *55*, 1–28. [CrossRef]
3. Dao, T.; Fu, D.Y.; Ermon, S.; Rudra, A.; Ré, C. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 16359.
4. Child, R.; Gray, I.; Radford, A.; Sutskever, I. Generating Long Sequences with Sparse Transformers. *arXiv* **2019**, arXiv:1904.10509.
5. Beltagy, I.; Peters, M.E.; Cohan, A. Longformer: The Long-Document Transformer. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), Virtual, 16–20 November 2020; pp. 288–298.
6. Zaheer, M.; Guruganesh, G.; Dubey, A.; Ainslie, J.; Alberti, C.; Ontañón, S.; Pham, P.; Ravula, A.; Wang, Q.; Yang, L.; et al. Big Bird: Transformers for Longer Sequences. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 17283–17297.
7. Wang, S.; Li, B.Z.; Khabsa, M.; Fang, H.; Ma, H. Linformer: Self-Attention with Linear Complexity. *arXiv* **2020**, arXiv:2006.04768.
8. Xiong, Y.; Zeng, Z.; Chakraborty, R.; Tan, M.; Fung, G.; Li, Y.; Singh, V. Nyströmformer: A Nyström-Based Algorithm for Approximating Self-Attention. *Proc. AAAI Conf. Artif. Intell.* **2021**, *35*, 14138–14148. [CrossRef]

9. Katharopoulos, A.; Vyas, A.; Pappas, N.; Fleuret, F. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. In Proceedings of the 37th International Conference on Machine Learning, PMLR, Virtual, 13–18 July 2020; Volume 119, pp. 5156–5165.
10. Choromanski, K.M.; Likhosherstov, V.; Dohan, D.; Song, X.; Gane, A.; Sarlós, T.; Hawkins, P.; Davis, J.Q.; Mohiuddin, A.; Kaiser, L.; et al. Rethinking Attention with Performers. In Proceedings of the International Conference on Learning Representations, Virtual, 3–7 May 2021.
11. Dao, T. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 3–7 May 2024.
12. Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; Guo, B. Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, 10–17 October 2021; pp. 10012–10022. [CrossRef]
13. Kitaev, N.; Kaiser, Ł.; Levskaya, A. Reformer: The Efficient Transformer. In Proceedings of the International Conference on Learning Representations (ICLR), Virtual, 26 April–1 May 2020.
14. Shen, Z.; Zhang, M.; Zhao, H.; Yi, S.; Li, H. Efficient Attention: Attention with Linear Complexities. *arXiv* **2018**, arXiv:1812.01243.
15. Fournier, Q.; Caron, G.M.; Aloise, D. A Practical Survey on Faster and Lighter Transformers. *ACM Comput. Surv.* **2023**, *55*, 1–40. [CrossRef]
16. Sun, Y.; Li, Z.; Zhang, Y.; Pan, T.; Dong, B.; Guo, Y.; Wang, J. Efficient Attention Mechanisms for Large Language Models: A Survey. *arXiv* **2025**, arXiv:2507.19595. [CrossRef]
17. Zhang, J.; Su, R.; Liu, C.; Wei, J.; Wang, Z.; Wang, H.; Zhang, P.; Jiang, H.; Huang, H.; Xiang, C.; et al. A Survey of Efficient Attention Methods: Hardware-efficient, Sparse, Compact, and Linear Attention. 2025; *preprint*.
18. Sukhera, J. Narrative Reviews: Flexible, Rigorous, and Practical. *J. Grad. Med. Educ.* **2022**, *14*, 414–417. [CrossRef] [PubMed]
19. Shah, J.; Bikshandi, G.; Zhang, Y.; Thakkar, V.; Ramani, P.; Dao, T. FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-Precision. *Adv. Neural Inf. Process. Syst.* **2024**, *37*, 68658–68685. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.