Approaching I/O-optimality for Approximate Attention

Pál András Papp* **Aleksandros Sobczyk** **Anastasios Zouzias**

Huawei Technologies Switzerland AG
Zurich, Switzerland

Abstract

We revisit the I/O complexity of *attention* in large language models. Given query–key–value matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d}$, and a machine with fast memory size M, the goal is to compute the "attention matrix" $\mathbf{A} = \mathrm{softmax}(\mathbf{Q}\mathbf{K}^\top/\sqrt{d})\mathbf{V}$ with the minimal number of data transfers between fast and slow memory. Existing methods in the literature, most notably FlashAttention and its variants, incur an I/O cost that depends quadratically on the context length n, while a trivial lower bound only requires $\Omega(nd)$ I/O's to read the inputs and write the output. In this work we show that the dependence on n can be reduced to almost-linear, by developing I/O-efficient algorithms inspired by the recent approximate attention method of Alman and Song [4, NeurIPS'23]. We also provide (algorithm-specific) lower bounds in different parameter regimes, which nearly match the corresponding upper bounds, indicating that the proposed algorithms are indeed close to I/O-optimal.

1 Introduction

Since its discovery [25], the attention mechanism has reshaped the entire field of machine learning, and, in particular, Large Language Models (LLMs). Given a sequence of tokens of length n, and a feature dimension d, the attention mechanism takes as input three matrices $\mathbf{Q} \in \mathbb{R}^{n \times d}$ (query), $\mathbf{K} \in \mathbb{R}^{n \times d}$ (key), and $\mathbf{V} \in \mathbb{R}^{n \times d}$ (value), and returns the *attention matrix*:

$$\mathbf{A} = \mathrm{Att}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) := \mathbf{D}^{-1} \exp(\mathbf{M})\mathbf{V}, \tag{1}$$

where $\mathbf{M} = \frac{1}{\sqrt{d}}\mathbf{Q}\mathbf{K}^\top$, $\exp(\mathbf{M})$ is element-wise exponentiation, and $\mathbf{D}$ is a diagonal matrix with $\mathbf{D}_{i,i} = \sum_{j=1}^{n} \exp(\mathbf{M}_{i,j})$ (in the literature $\mathbf{D}^{-1}\exp(\mathbf{M})$ is often written as $\mathrm{softmax}(\frac{1}{\sqrt{d}}\mathbf{Q}\mathbf{K}^\top)$).

The straightforward algorithm, which first computes the matrix $\mathbf{Q}\mathbf{K}^\top$ using standard matrix multiplication, has an arithmetic complexity of $O(n^2 d)$. In the so-called "long-context" regime, where n can grow very large, the quadratic complexity often forms a computational bottleneck. Overcoming the $O(n^2)$ cost of attention is one of the most interesting computational challenges of the last decade. Indeed, the literature on fast and efficient attention algorithms is quite extensive, where common techniques to reduce the arithmetic intensity include approximation algorithms [8, 11, 15, 18, 26], sparsification [7, 27], or replacing the softmax with a kernel function [17]. A particularly important work on approximate attention is the recent algorithm by Alman and Song [4], which also comes with strong theoretical bounds on the approximation guarantee.

The aforementioned works can considerably reduce the arithmetic complexity of attention. However, in modern systems, the computational runtimes also depend on several other aspects. One particularly important factor is the amount of data movements within the memory hierarchy. Due to this, there is a line of research analyzing computations with respect to their *I/O complexity* or I/O cost, i.e. the data movement they require when executed in a two-level memory hierarchy with a fast memory of

*Contacts: pal.andras.papp@huawei.com, aleksandros.sobczyk@huawei.com, anastasios.zouzias@huawei.com

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

limited capacity M (e.g. cache) and a slow memory of unlimited capacity (e.g. RAM). For attention, if we simply use the I/O-optimal method for standard matrix multiplication, we obtain an I/O cost of $O(\frac{n^2 \cdot d}{\sqrt{M}})$. However, the celebrated FlashAttention algorithm by Dao et al [10], and follow-up works [9, 23], allow to execute the same algorithm with an I/O cost of only $O(\frac{n^2 \cdot d^2}{M})$ by fusing the two multiplications and re-arranging the order of execution. This is indeed an improvement over the naive method when $M > d^2$, i.e. the cache size is sufficiently large, which is often the case in practice. The work of Saha and Ye [22] also proves tight (conditional) lower bounds for both of these cases up to constant factors, showing that these algorithms (naive matrix multiplication and FlashAttention) are indeed I/O-optimal in the corresponding cache size regimes.

In this work, we combine the two lines of research above: we study the most I/O-efficient way to execute attention if we use the state-of-the-art algorithm from [4]. Our results show that this approximate attention algorithm can significantly outperform classical attention not only in terms of computational, but also in terms of I/O complexity. In particular, its asymptotic I/O cost is also significantly lower than that of the FlashAttention algorithm for classical attention.

1.1 Contributions

Our main results are tight and nearly-tight bounds for the I/O complexity of attention when computed using the algorithm of Alman and Song [4]. It turns out to that this I/O complexity can behave rather differently depending on the relations between several parameters: the feature dimension d of the input matrices, the polynomial degree g used to approximate the exponential function in [4], the size $r = \binom{d+g}{g}$ of the approximation matrices in [4], and the fast memory capacity M. For clarity, the role of these parameters is summarized in Table 1. We distinguish the following cases:

- **Case I:** when $M = \Omega(d \cdot r)$, i.e. the fast memory is so large that it can essentially fit all of intermediate matrix $\mathbf{U}_2^\top \mathbf{V}$ from [4], up to constant factors;
- **Case II:** when $M = o(d \cdot r)$, and also $g = o(\log M)$, i.e. the fast memory is more than exponentially larger than the degree g of the approximating polynomial;
- **Case III:** when $M = o(d \cdot r)$, and $g = \Omega(\log M)$ but $g = o(\sqrt{M})$, i.e. the fast memory is between exponential and quadratic in the degree g;
- **Case IV:** when $M = o(d \cdot r)$ and $g = \Omega(\sqrt{M})$, i.e. the fast memory is at most quadratic in g.

We prove different I/O bounds for these cases. In Case I, when the fast memory essentially fits the intermediate matrix $\mathbf{U}_2^\top \mathbf{V}$ in [4], a simpler strategy produces tight bounds up to a constant factor.

Theorem 1.1 (Case I.)**.** *If $M = \Omega(d \cdot r)$, then the optimal I/O complexity of Approximate Attention is both upper and lower bounded by $\Theta(n \cdot d)$.*

In Case II, when the degree of the approximating polynomial is small (compared to the fast memory size), the upper bounds we obtain are somewhat more complex.

Theorem 1.2 (Case II.)**.** *If $M = o(d \cdot r)$ and $g = o(\log M)$, then the optimal I/O complexity of Approximate Attention is upper bounded by*

$$O\left(\frac{n \cdot r \cdot d \cdot (4e^2)^g}{M^{\frac{g}{g+1}}}\right),$$

and lower bounded by

$$\Omega\left(\frac{n \cdot r \cdot d \cdot g}{M^{\frac{g}{g+1}}}\right).$$

When the polynomial degree is larger or fast memory capacity is small, the problem becomes even more technical. In these cases we need an additional assumption $d \geq 5g$ for our lower bounds.

Theorem 1.3 (Case III.)**.** *If $M = o(d \cdot r)$ and $\Omega(\log M) \leq g \leq o(\sqrt{M})$, then the optimal I/O complexity of Approximate Attention is upper bounded by*

$$O\left(\min\left(\frac{n \cdot r \cdot d^2}{M}, \frac{n \cdot r \cdot d}{\sqrt{M}}\right)\right),$$

and when $d \geq 5g$, it is also lower bounded by $\Omega\left(\frac{n \cdot r \cdot d \cdot g}{M}\right)$.

Table 1: Notation and summary of the main parameters used throughout our paper.

Parameter	Origin	Description
n (sequence length)	Input problem	Number of rows in $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$.
d (feature dimension)	Input problem	Number of columns in $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$. Often assumed to be $O(\log n)$ for long-context attention [4].
M (fast memory capacity)	Architecture	The amount of available fast memory (i.e. cache size), which determines the optimal I/O cost.
g (degree of approximating polynomial)	Algorithm of Alman & Song	Degree of the polynomial used to approximate the exponentiation. Depends on the accepted approximation error and the magnitude of the entries. Assumed to be $o(\log n)$ in [4].
r (dimension of the approximating matrices)	Algorithm of Alman & Song	The maximal number of terms in a polynomial of degree g on d variables. Its value equals $\binom{d+g}{g}$. It determines the number of columns in the approximation matrices $\mathbf{U}_1$ and $\mathbf{U}_2$.
w (generating set size)	Our I/O analysis	The number of entries we decide to load from slow memory in a row of $\mathbf{Q}$ (or $\mathbf{K}$), used in our proofs. If $w \leq d$, this allows to compute at most $\binom{w+g}{g}$ entries in a row of $\mathbf{U}_1$ (or $\mathbf{U}_2$).

Finally, for tiny cache sizes $M = O(g^2)$, the problem behaves like standard matrix multiplication.

Theorem 1.4 (Case IV.)**.** *If $M = o(d \cdot r)$ and $g = \Omega(\sqrt{M})$, the optimal I/O complexity of Approximate Attention is upper and lower bounded by $\Theta\left(\frac{n \cdot r \cdot d}{\sqrt{M}}\right)$, where the lower bound also requires $d \geq 5g$.*

For convenience, we summarize the results in Table 2. Theorems 1.1-1.4 provide a comprehensive overview of the optimal I/O complexity of Approximate Attention. From a broader perspective, our results can also be understood as an I/O analysis of matrix multiplication in a special setting where one of the input matrices is sparse, or behaves unusually in some way regarding I/O. We believe that out proof techniques could also inspire further results in sparse attention computation or in entirely different domains.

Table 2: Summary of the reported I/O complexity bounds.

	Regime	**Upper Bound**	**Lower Bound**
Thm.1.1 (Case I)	$M = \Omega(dr)$	$O(nd)$	$\Omega(nd)$
Thm.1.2 (Case II)	$M = o(dr)$ $g = o(\log M)$	$O\left(\frac{nrd(4e^2)^g}{M^{\frac{g}{g+1}}}\right)$	$\Omega\left(\frac{nrdg}{M^{\frac{g}{g+1}}}\right)$ †
Thm.1.3 (Case III)	$M = o(dr)$ $\Omega(\log M) \leq g \leq o(\sqrt{M})$	$O\left(\min\left(\frac{nrd^2}{M}, \frac{nrd}{\sqrt{M}}\right)\right)$	$\Omega\left(\frac{nrdg}{M}\right)$ †
Thm.1.4 (Case IV)	$M = o(dr)$ $g = \Omega(\sqrt{M})$	$O\left(\frac{nrd}{\sqrt{M}}\right)$	$\Omega\left(\frac{nrd}{\sqrt{M}}\right)$ †

† The lower bounds in Cases II, III, and IV are obtained using S-partitions and they are algorithm-specific. All other bounds are algorithm-agnostic.

1.2 Parameters and discussion

We discuss the bounds in more detail with some example parameters for clarity.

For the polynomial degree g, Alman and Song [4] consider $g = o(\log n)$, which holds, for example, when $B = o(\log n)$ and $\epsilon = 1/n$; together with $d = O(\log n)$, this is already enough to establish $r = n^{o(1)}$ in their analysis. In practice, however, B and ϵ may not grow as a function of the context length n at all. Instead, for long–context attention, it is reasonable to treat these parameters independently, or even as constants.

If $g = O(1)$, then the upper and lower bounds in Theorems 1.2 and 1.3 are also tight up to a constant factor. This is easy to see for Theorem 1.2. In Theorem 1.3, the gap between the bounds is at most $\frac{\sqrt{M}}{g}$; with $g = \Omega(\log M)$, we can upper bound this by $2^{O(g)}$, which is again a constant. Theorems 1.1 and 1.4 are less interesting from this angle: these bounds are always tight.

In order to compare the bounds to FlashAttention, let us consider $d^2 < M < n \cdot d$; this is the range where FlashAttention improves upon naive matrix multiplication [22].

For Case I, assume the aforementioned regime from [4], namely, $d = O(\log(n))$, $B = o(\sqrt{\log(n)})$ and $\epsilon = 1/n$, which ensures that $r = n^{o(1)}$. The condition of Case I is $M = \Omega(d \cdot r)$; hence we are in this setting if M is larger than $d \cdot r = n^{o(1)}$. Theorem 1.1 reports an $O(n \cdot d)$ upper bound, which matches the trivial lower bound of $\Omega(n \cdot d)$, while FlashAttention incurs a cost of $\Theta(\frac{n^2 \cdot d^2}{M})$ [22]. For all $M \in \Omega(d \cdot r) \cap o(n \cdot d)$, the latter becomes $\omega(n \cdot d)$, which implies a strict asymptotic separation. For example, for $M = n^{0.1} \cdot d$, our upper bound remains $O(n \cdot d)$, but the I/O complexity of FlashAttention is $\Theta(n^{1.9} \cdot d)$. The factor of difference $\frac{n \cdot d}{M}$ between our method and FlashAttention becomes more significant as n grows.

For the remaining cases, the bounds we obtain are again almost-linear in n: for Cases II, III and IV, respectively, we get upper bounds of

$$O\left(\frac{n^{1+o(1)} \cdot d}{M^{\frac{g}{g+1}}}\right), \quad O\left(\min\left(\frac{n^{1+o(1)} \cdot d^2}{M}, \frac{n^{1+o(1)} \cdot d}{\sqrt{M}}\right)\right) \quad \text{and} \quad O\left(\frac{n^{1+o(1)} \cdot d}{\sqrt{M}}\right).$$

In particular, the first two of these expressions are at most $O\left(\frac{n^{1+o(1)} \cdot d^2}{M}\right)$ in the given parameter regimes. FlashAttention has a very similar I/O cost of $\Theta(\frac{n^2 \cdot d^2}{M})$, which depends quadratically on n; hence compared to it, our algorithms save at least an almost-linear factor of I/O cost in n.

Finally, we make a remark regarding the exponential gap (with respect to g) in Case II. Note that in that case $g = o(\log(M))$. This implies $(4e^2)^g = M^{o(1)}$, and, therefore, the corresponding gap between the upper and lower bound is in fact a factor of $M^{o(1)}$. Theorem 1.2 explicitly states the dependence on g for completeness, but this observation can be used to simplify the bounds if required.

2 Preliminaries

2.1 Approximate attention

Many existing works on attention algorithms target the *exact* computation of the attention matrix. While this is not necessarily "unreasonable", it raises some intricate questions. In particular, if attention can be computed exactly in T steps, then $\exp(x)$ can be also computed exactly in $T + O(1)$ steps (see Appendix D.1). However, assuming that the entire sequence of digits of $\exp(x)$ can be returned in a single "time-step" might raise concerns regarding the machine model; see e.g. the discussion in [14]. To avoid such intricacies, we target *approximate attention* algorithms, rather than exact ones. We consider the following approximate attention definition from [4].

Problem 2.1 (Additive-error Approximate Attention [4])**.** *Given $\epsilon > 0$, and matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n \times d}$, return a matrix $\widetilde{\mathbf{A}}$ which satisfies:*

$$\max_{i,j} \left| \widetilde{\mathbf{A}}_{i,j} - \mathrm{Att}(\mathbf{Q}, \mathbf{K}, \mathbf{V})_{i,j} \right| \leq \epsilon.$$

This specific definition has proven useful to obtain sharp bounds on the arithmetic complexity of attention. When $d = O(\log(n))$ and $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ have entries of magnitude $o(\sqrt{\log(n)})$, the authors of [4] describe an algorithm with nearly-linear $n^{1+o(1)}$ complexity in the algebraic model; we describe this in detail in Section 2.2. On the other hand, the authors also prove that if the magnitude of entries in $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ is $\Theta(\sqrt{\log(n)})$, then no algorithm can solve Problem 2.1 up to $1/\operatorname{poly}(n)$ error in sub-quadratic time under the Strong Exponential Time Hypothesis (SETH).

2.2 The algorithm of Alman and Song

In brief, the algorithm of Alman and Song [4] that solves 2.1 works as follows:

1. Compute the coefficients of a degree-g polynomial $P(x)$, which approximates $\exp(x)$ in the domain $[-B, B]$, where B is a bound on the magnitudes of the elements of $\mathbf{Q}, \mathbf{K}$.
2. Construct two matrices $\mathbf{U}_1, \mathbf{U}_2 \in \mathbb{R}^{n \times r}$ for some $r = n^{o(1)}$, such that $\mathbf{U}_1 \mathbf{U}_2^\top = P(\mathbf{Q}\mathbf{K}^\top) \approx \exp(\mathbf{Q}\mathbf{K}^\top / \sqrt{d})$. Intuitively, the i^{th} row of $\mathbf{U}_1$ (i^{th} row of $\mathbf{U}_2$, respectively) is obtained from the i^{th} row of $\mathbf{Q}$ (i^{th} row of $\mathbf{K}$), via an expansion of the polynomial $P(\cdot)$.
3. Return $\widetilde{\mathbf{A}} \leftarrow \widetilde{\mathbf{D}}^{-1}(\mathbf{U}_1(\mathbf{U}_2^\top \mathbf{V}))$, where $\widetilde{\mathbf{D}} = \mathrm{diag}(\mathbf{U}_1(\mathbf{U}_2^\top \mathbf{1}))$.

The key idea that makes this algorithm efficient is that the polynomial approximation completely removes the non-linearity of the $\mathrm{softmax}$. This allows to execute the matrix multiplication $\mathbf{U}_2^\top \mathbf{V}$ first, and then $\mathbf{U}_1(\mathbf{U}_2^\top \mathbf{V})$ afterwards, which altogether needs significantly fewer arithmetic operations.

Going in more detail, in step 2 we have that $\exp(\mathbf{Q}\mathbf{K}^\top)_{i,j} = \exp(\mathbf{q}^\top \mathbf{k})$, where $\mathbf{q}^\top$ is the i^{th} row of $\mathbf{Q}$ and $\mathbf{k}$ is the j^{th} column of $\mathbf{K}^\top$. When $\exp(x)$ is replaced by the polynomial $P(x)$, we can write:

$$P(\mathbf{q}^\top \mathbf{k}) = P(q_1 k_1 + q_2 k_2 + \ldots + q_d k_d) = \sum_{l=0}^{g} c_l (q_1 k_1 + q_2 k_2 + \ldots + q_d k_d)^l.$$

If we expand $(q_1 k_1 + q_2 k_2 + \ldots + q_d k_d)^l$ for any $l \in \{1, \ldots, g\}$, we obtain a sum with $\binom{d+l-1}{l}$ additive terms. Each additive term is of the form $(q_1 k_1)^{l_1} \cdot (q_2 k_2)^{l_2} \cdot \ldots \cdot (q_d k_d)^{l_d}$, where $l_1 + l_2 + \ldots + l_d = l$. Altogether, we get $r = \sum_{l=0}^{g} \binom{d+l-1}{l}$ additive terms. Ultimately, since each term is the product of some powers of the elements of $\mathbf{q}$ and $\mathbf{k}$, the polynomial $P(\mathbf{q}^\top \mathbf{k})$ can be written as a bilinear form $P(\mathbf{q}^\top \mathbf{k}) = \mathbf{u}^\top \mathbf{C} \mathbf{v}$, where $\mathbf{u} \in \mathbb{R}^r$ consists of products of (powers of) the elements of $\mathbf{q}$, while $\mathbf{v} \in \mathbb{R}^r$ consists of products of (powers of) the elements of $\mathbf{k}$, and $\mathbf{C} \in \mathbb{R}^{r \times r}$ is a diagonal matrix with scalar coefficients that depend on the polynomial $P(\cdot)$, but are independent from n.

To ensure $|P(x) - \exp(x)| \leq \epsilon \cdot \exp(x)$, setting $g := \Theta\left(\max\left\{\frac{\log(1/\epsilon)}{\log(\log(1/\epsilon)/B^2)}, B^2\right\}\right)$ is sufficient [1, 4]. The coefficients of $P(\cdot)$ can be computed as $\mathrm{poly}(g)$-bit rationals in $\mathrm{poly}(g)$ time.

Below we introduce a specific notation for the number of possible terms in a polynomial of degree g on w variables; this will play a central role in our analysis.

Definition 2.1. *Let us define the function $\tau : \mathbb{Z}^+ \to \mathbb{Z}^+$ as a shorthand notation for*

$$\tau(w) := \sum_{l=0}^{g} \binom{w+l-1}{l} = \binom{w+g}{g}.$$

Intuitively, $\binom{w+l-1}{l}$ is a combination with repetition, i.e. the number of ways we can divide a total degree of g among w variables. The number of columns in $\mathbf{U}_1$ and $\mathbf{U}_2$ is $r = \tau(d)$. The closed form expression of the sum in the definition can be shown through an induction; we defer the proof to Appendix D. We note that the work of [4] uses a looser bound of $\binom{2(w+g)}{2g}$ to bound r.

Note that the algorithm above executes matrix multiplications with a standard inner product-based algorithm. Previous works on the I/O analysis of attention also focus on standard matrix multiplication [22]. It is well-known that "Strassen-like", fast matrix multiplication algorithms (see e.g. [3, 5] and references therein) can achieve better computational and I/O complexity [12]. However, these algorithms are often known to be rather impractical, and their analysis is more complex than the standard algorithm, especially when rectangular matrices are involved.

Throughout the paper, we analyze I/O-efficient algorithms based on the aforementioned Approximate Attention method. Note that our upper bounds naturally carry over to solving Problem 2.1 in general. On the other hand, lower bounds in standard I/O-complexity models are inherently coupled to a concrete algorithm, so our bounds here are specific to this Approximate Attention technique.

2.3 Model of computation

The most prominent model to analyze the I/O complexity of computations is the red-blue pebble game of Hong and Kung [16]. In this setting, a computation is captured as a directed acyclic graph (DAG), where the nodes represent operations, and the directed edges represent data dependencies, i.e.

that the output of an operation is required as an input for another operation. The computational DAG corresponding to the approximate attention algorithm from [4] is illustrated in Figure 1. Note that the algorithm only uses basic algebraic operators $\{+, -, \times, /\}$, and square roots, hence avoiding the intricacies of exact exponentiation that we mentioned in Section 2.1.

The red-blue pebble game models the execution of the computation in a two-level memory hierarchy: we have a fast memory with limited capacity M, and slow memory of unlimited capacity. If the output of an operation is currently stored in fast (slow) memory, then this is indicated by having a red (blue) pebble on the node. The number of red pebbles can not exceed the cache capacity M at any time. Loading a value from (respectively, saving a value to) slow memory then corresponds to placing a red (blue) pebble on a node that already has a blue (red) pebble. A new output value can only be computed (placing a red pebble on the node) if all the parents of the node have a red pebble.

The execution of the computation is modeled by a sequence of steps, where each step can load, save, compute or delete a value. Initially, the source nodes of the DAG have a blue pebble, and the process finishes when all the sink nodes have a blue pebble. Since the goal of the model is to capture I/O complexity, the cost of the pebbling sequence is the total number of save and load operations executed; compute and delete steps are considered free.

Red-blue pebbling has been thoroughly analyzed in several works, and widely used to establish upper and lower bounds on the I/O cost of specific computations [6, 13, 16, 19, 20, 21, 24].

3 The I/O Cost of Approximate Attention

In this section, we outline the main intuition behind the bounds in Theorems 1.1-1.4, and the main ingredients of the proofs. The detailed proofs are deferred to Appendices A, B and C.

In our I/O analysis, we focus on the dominant step of computing $\mathbf{U}_1(\mathbf{U}_2^\top \mathbf{V})$; the other parts of the computation are negligible compared to this. $\mathbf{U}_1(\mathbf{U}_2^\top \mathbf{1})$ can be computed by executing a further column to $\mathbf{V}$, which does not increase the cost asymptotically. The normalization step is executed on the output matrix of size $n \times d$; even if each entry is naively re-loaded and saved, this only adds $O(n \cdot d)$, which is a lower bound on the cost in any case.

The computation $\mathbf{U}_1(\mathbf{U}_2^\top \mathbf{V})$ consists of two consecutive matrix multiplications; we denote the intermediate matrix by $\mathbf{H} := \mathbf{U}_2^\top \mathbf{V}$. Throughout the proofs, we focus on the second multiplication $\mathbf{U}_1\mathbf{H}$; note that $\mathbf{U}_1 \in \mathbb{R}^{n \times r}$ and $\mathbf{H} \in \mathbb{R}^{r \times d}$. The lower and upper bounds are then easy to extend to the entire computation $\mathbf{U}_1(\mathbf{U}_2^\top \mathbf{V})$. Considering the sub-problem $\mathbf{U}_1\mathbf{H}$ separately is in fact a restriction of I/O strategies: the entries in $\mathbf{H}$ here are all inputs that need to be loaded from slow memory. This rules out "fusion-based" strategies that never save the entries of $\mathbf{H}$, but instead immediately go on to multiply them with $\mathbf{U}_1$. Intuitively, these strategies can be ignored because $\mathbf{H}$ is smaller than $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$, so saving/loading it does not affect the magnitude of the total cost.

3.1 Intuition: optimal tile design

In standard matrix multiplication, the key to optimal I/O strategies is to divide the matrix into rectangular tiles to maximize reusing the same inputs and outputs. In general, when multiplying matrices $\mathbf{A} \in \mathbb{R}^{m_1 \times m_2}$ and $\mathbf{B} \in \mathbb{R}^{m_2 \times m_3}$, the optimal strategy uses square tiles. That is, we read the inputs in a square-shaped $\Theta(\sqrt{M}) \times \Theta(\sqrt{M})$ tile in $\mathbf{A}$, and another square-shaped $\Theta(\sqrt{M}) \times \Theta(\sqrt{M})$ tile in $\mathbf{B}$, and we execute the corresponding part of the matrix multiplication. This gives us a partial sum for each entry in a $\Theta(\sqrt{M}) \times \Theta(\sqrt{M})$ tile of the output matrix, which we can add (and save) to the already aggregated values for this specific output entry. Each such tiling step only requires a fast memory capacity of $\Theta(M)$, it has an I/O cost of only $\Theta(M)$, and covers $\Theta(\sqrt{M})^3 = \Theta(M^{3/2})$ of the scalar multiplications that compose the entire operation. With a total of $m_1 \cdot m_2 \cdot m_3$ multiplications in the entire operation, this allows for an algorithm with an I/O cost of $\Theta\left(\frac{m_1 \cdot m_2 \cdot m_3}{\sqrt{M}}\right)$. This general tiling idea is illustrated in Figure 2 for the matrix multiplication $\mathbf{U}_1\mathbf{H}$.

Intuitively speaking, this tiling is optimal for standard matrix multiplication because it allows to execute the highest number $\Theta(M^{3/2})$ of the scalar multiplications while the tile sizes in the input/output matrices is still $\Theta(M)$. The best such tiling is naturally obtained when we use square-shaped

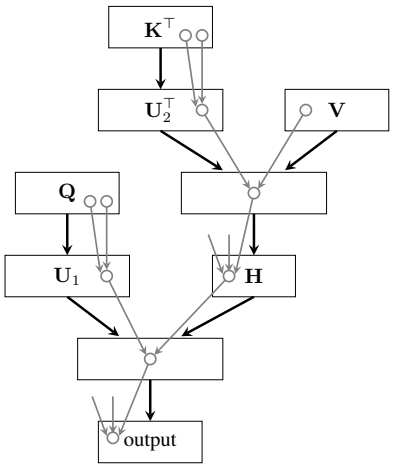


Figure 1: Sketch of the computational DAG for the approximate attention algorithm [4]. Each box represents a set of nodes. In matrix multiplication, we first multiply pairs of entries from the input matrices, and then sum these up to from an entry in the output. In $\mathbf{U}_1$ (and $\mathbf{U}_2$), each node is computed from at most g nodes of $\mathbf{Q}$ (and $\mathbf{K}$).

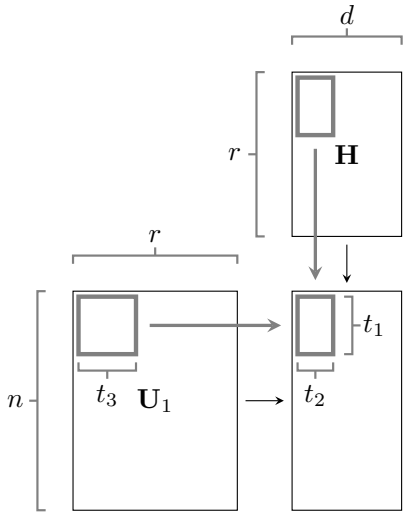


Figure 2: Illustration of matrix multiplication with tiles, for the example $\mathbf{U}_1\mathbf{H}$. The output matrix is split to tiles of size $t_1 \times t_2$, and each of these is aggregated in tiles of width t_3 (respectively, height t_3) in the corresponding row strip of $\mathbf{U}_1$ (column strip of $\mathbf{H}$). Altogether, the output matrix is computed via $\frac{n}{t_1} \cdot \frac{d}{t_2} \cdot \frac{r}{t_3}$ tiling steps.

tiles in the inputs and the output. However, with our special matrices $\mathbf{U}_1$ and $\mathbf{U}_2$, the situation is significantly different. The entries in a given row of $\mathbf{U}_1$ are not inputs, but values computed from the corresponding row of $\mathbf{Q}$; in fact, by loading only w entries from the i^{th} row of $\mathbf{Q}$ (for some $w \leq d$), we can compute up to $\tau(w) = \binom{g+w}{g}$ different entries in the i^{th} row of $\mathbf{U}_1$.

This means that in order to form an essentially square-shaped tile in $\mathbf{U}_1$ with $\Theta(M)$ load operations, we actually need to solve the following equation for w:

$$\tau(w) = \frac{\Theta(M)}{w}. \tag{2}$$

Let us denote the solution of this by w^*. We can then select $\frac{\Theta(M)}{w^*}$ rows of $\mathbf{U}_1$, and load w^* values in the corresponding rows of $\mathbf{Q}$, thus generating up to $\tau(w^*)$ entries in each row of $\mathbf{U}_1$.

Forming tiles of this shape is the key idea behind our results. Intuitively, Case II marks the general case where this choice of w^* is indeed possible. The other settings can be be understood as special cases. In Case I, we essentially have $w^* \geq d$; since we cannot load more than d values in a row of $\mathbf{Q}$, we can only form 'taller' rectangular tiles. In Cases III and IV, we have $w^* \leq g$, but we still need to load g values from $\mathbf{Q}$ in many cases because many degree-g terms in the polynomial actually contain g different variables; hence we need to form 'wider' rectangular tiles.

Note that obtaining a closed-form expression for w^* from Equation (2) is non-trivial; instead, we rely on standard bounds on binomial coefficients. Specifically, we use the fact that $\left(\frac{a}{b}\right)^b \leq \binom{a}{b} \leq \left(\frac{e \cdot a}{b}\right)^b$.

3.2 Generic upper bounds

Note that if we naively apply the standard matrix multiplication method with tiles of size $\frac{\sqrt{M}}{4} \times \frac{\sqrt{M}}{4}$ on our computation, we get an algorithm of I/O cost $O(\frac{n \cdot r \cdot d}{\sqrt{M}})$. This is indeed viable when $\frac{\sqrt{M}}{4} \leq d$, i.e. each of the matrices can fit the square tiles. When $\frac{\sqrt{M}}{4} > d$, we can instead form tiles of size $\frac{M}{4d} \times d$ in both $\mathbf{H}$ and the output matrix, and tiles of size $\frac{M}{4d} \times \frac{M}{4d}$ in $\mathbf{U}_1$ (assuming that these tiles fit, i.e. we have $\frac{M}{4d} \leq r$). Each of these tiles only needs $\frac{M}{4}$ I/O operations, because in $\mathbf{U}_1$, we

can obtain all the entries of a row with only d I/O steps. The number of tiling steps is altogether $\frac{n}{\frac{M}{4d}} \cdot \frac{r}{\frac{M}{4d}} = O(\frac{nrd^2}{M^2})$. With an I/O cost of $O(M)$ per tile, we altogether get $O(\frac{nrd^2}{M})$ I/O steps.

Note that $\frac{n \cdot r \cdot d}{\sqrt{M}} \leq \frac{nrd^2}{M}$ exactly when $\sqrt{M} \leq d$, so the two bounds above can be easily combined. This gives a generic upper bound that holds over Cases II, III and IV.

Lemma 3.1. *If $M = o(d \cdot r)$, the optimal I/O cost of Approximate Attention is upper bounded by*

$$O\left(\min\left(\frac{n \cdot r \cdot d^2}{M}, \frac{n \cdot r \cdot d}{\sqrt{M}}\right)\right).$$

3.3 Case I: proof of Theorem 1.1

In Case I, we have $d \cdot \tau(d) = d \cdot r = O(M)$, i.e. there is a constant $0 < c_0 < 1$ such that $c_0 \cdot d \cdot r \leq \frac{1}{4}M$. This means that we can select $\frac{M}{4 \cdot d}$ rows from $\mathbf{U}_1$, load all the d values in the corresponding rows of $\mathbf{Q}$ (at an I/O cost of $\frac{1}{4}M$), and generate all the r entries in each of these rows in $\mathbf{U}_1$.

For the computation, we can then split the output matrix into tiles of height $\frac{M}{4 \cdot d}$ and width $c_0 \cdot d$. We can compute each tile in a single iteration: we create the corresponding $\frac{M}{4 \cdot d}$ rows in $\mathbf{U}_1$ as discussed, we load the tile of shape $r \times (c_0 \cdot d)$ from $\mathbf{H}$ that needs to multiply it (at an input cost of at most $\frac{1}{4}M$), and then output the final result on the tile (output cost of at most $\frac{1}{4}M$). In each tile, all the loaded entries in $\mathbf{Q}$ and $\mathbf{H}$ and the aggregated values in the output matrix can be kept in fast memory simultaneously (since $\frac{3}{4}M < M$) for the entire duration of computing the tile. On the other hand, the entries of $\mathbf{U}_1$ are always recomputed from the entries in $\mathbf{Q}$, without requiring any new I/O steps. The number of tiles needed to cover the output matrix is $\frac{n}{M/(4d)} \cdot \frac{d}{c_0 \cdot d} = O(\frac{n \cdot d}{M})$, altogether resulting in an I/O cost of $O(n \cdot d)$.

The first part of the computation, i.e. the other matrix multiplication $\mathbf{H} = \mathbf{U}_2^\top \mathbf{V}$, can be executed in a similar way at the same I/O cost. Here d entries in a row of $\mathbf{K}$ allow us to generate the entire column of $\mathbf{U}_2^\top$, so we can form tiles of shape $r \times \frac{M}{4 \cdot d}$ in $\mathbf{U}_2^\top$, $\frac{M}{4 \cdot d} \times (c_0 \cdot d)$ in $\mathbf{V}$ and $r \times (c_0 \cdot d)$ in $\mathbf{H}$, aggregating each tile in the output matrix in $\frac{n}{M/(4d)}$ steps.

Note that the corresponding lower bound is straightforward: each entry of the input matrices $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ must be loaded at least once, giving an I/O cost of at least $3 \cdot n \cdot d$.

3.4 Case II: when $g \leq w^* < d$

In Case II, the solution to Equation (2) satisfies $g \leq w^* < d$. The best strategy here, as discussed above, is to tile $\mathbf{U}_1$ by loading $w = w^*$ values from $\frac{\Theta(M)}{w}$ distinct rows of $\mathbf{Q}$. Consider a specific row $\mathbf{q}$ of $\mathbf{Q}$, and let $\mathbf{u}$ be the corresponding row in $\mathbf{U}_1$. Hypothetically, if every set of w loaded values from $\mathbf{q}$ could generate $\tau(w)$ *distinct* entries in $\mathbf{u}$, then the strategy would provide matching upper and lower bounds. In this case, the corresponding tiles in $\mathbf{H}$ and $\mathbf{U}_1\mathbf{H}$ could both have height $\frac{\Theta(M)}{w}$ and width $\Theta(w)$, so all inputs and outputs in a tile would fit into fast memory. This altogether gives $\Theta\left(\frac{n}{\frac{M}{w}} \cdot \frac{d}{w}\right)$ tiles in the output matrix, each aggregated from $\Theta(\frac{r}{\frac{M}{w}})$ separate column strips in $\mathbf{U}_1$ and row strips in $\mathbf{H}$. The total number of tiling steps would be $\Theta(\frac{nd}{M} \cdot \frac{rw}{M}) = \Theta(\frac{ndrw}{M^2})$. With $\Theta(M)$ I/O steps per tile, the total cost is $\Theta(\frac{ndrw}{M})$. In Appendix A, we show that no I/O strategy can be more efficient than this. The proof analyzes the so-called S-partitions of the computational graph, which is a long-established tool to derive I/O lower bounds on different computations [16, 20].

In contrast to this hypothetical setting, the w loaded values *cannot* generate fully-disjoint sets of entries from $\mathbf{u}$ in each tile. To ensure that every entry of $\mathbf{u}$ is obtainable in at least one tile, the entries of $\mathbf{q}$ actually need to be loaded in multiple different tiles. This means that there will be many entries in $\mathbf{u}$ that can be generated in multiple tiles. We can assign each such entry of $\mathbf{u}$ to an arbitrary one of these tiles, but this still implies that the width of the average tile in $\mathbf{U}_1$ will be lower than $\tau(w)$.

As such, we need to select different subsets of w values each from $\mathbf{q}$, and assign each entry in $\mathbf{u}$ to one of these subsets that can generate it. For this, we divide the d entries of $\mathbf{q}$ into $\frac{d \cdot g}{w}$ groups of size $\frac{w}{g}$ each, and consider all the $\binom{\frac{d \cdot g}{w}}{g}$ possible combinations of g different groups as generator sets for a

tile. All entries in $\mathbf{u}$ can be obtained in at least one combination, since they are generated from at most g entries in $\mathbf{q}$. For entries in $\mathbf{u}$ that can be generated in multiple tiles, we simply assign it to the first such tile, hence some of our tiles in $\mathbf{U}_1$ will be wider than others.

Altogether, each tile of the output matrix is aggregated through $\binom{\frac{d\cdot g}{w}}{g}$ steps. With tiles of size $\frac{\Theta(M)}{w} \times \Theta(w)$ in an output matrix of dimensions $n \times d$, and an I/O cost of $O(M)$ per tile, this gives a total I/O cost of $\frac{n}{\frac{\Theta(M)}{w}} \cdot \frac{d}{\Theta(w)} \cdot \binom{\frac{d\cdot g}{w}}{g} \cdot O(M) = O\left(n \cdot d \cdot \binom{\frac{dg}{w}}{g}\right)$.

Formally, the observations above can be expressed through the following lemma.

Key Lemma 3.2. *Let us choose an integer w such that $g \leq w \leq d$ and $w \cdot \tau(w) \leq \frac{1}{4}M$. Then the optimal I/O cost of Approximate Attention in Case II is upper bounded by*

$$O\left(n \cdot d \cdot \binom{\frac{dg}{w}}{g}\right),$$

and lower bounded by

$$\Omega\left(\frac{n \cdot r \cdot d \cdot w}{M}\right).$$

The proof is provided in Appendix A. Theorem 1.2 then follows from this more general statement.

Proof of Theorem 1.2. Given Key Lemma 3.2, the bounds in Theorem 1.2 are obtained by choosing

$$w_0 = \frac{1}{4 \cdot e} \cdot g \cdot M^{\frac{1}{g+1}}$$

for w. Note that this w_0 is only slightly different from the real w^*, due to only approximating the binomial coefficient in $\tau(w)$. First note that $w_0 \geq g$; this is equivalent to $M \geq (4e)^{g+1}$, which indeed holds if $g = o(\log M)$. We then show that w_0 satisfies $w_0 \cdot \tau(w_0) \leq \frac{1}{4}M$. Indeed, we have

$$w_0 \cdot \tau(w_0) = w_0 \cdot \binom{w_0 + g}{g} \leq w_0 \cdot \left(\frac{2 \cdot e \cdot w_0}{g}\right)^g$$

using the upper bound on the binomial coefficient, and $w_0 \geq g$. Substituting w_0 into this, we get $\frac{1}{4e} \cdot \frac{g}{2^g} \cdot M$. This is indeed smaller than $\frac{1}{4}M$, since $g < 2^g$ for any positive integer g. From this, it also follows that $w_0 \leq d$, since in Case II, we have $d \cdot \tau(d) = \omega(M)$.

Using $w = w_0$ in the lower bound of Key Lemma 3.2 directly gives our lower bound in Theorem 1.2:

$$\Omega\left(\frac{n \cdot r \cdot d}{M} \cdot w_0\right) = \Omega\left(\frac{n \cdot r \cdot d \cdot g}{M^{\frac{g}{g+1}}}\right).$$

Similarly, the upper bound of Key Lemma 3.2 provides the upper bound in Theorem 1.2 if we substitute $w = w_0$, and use $\binom{\frac{dg}{w}}{g} \leq (\frac{edg}{wg})^g$ and $r = \binom{d+g}{g} \geq (\frac{d}{g})^g$:

$$O\left(n \cdot d \cdot \binom{\frac{dg}{w_0}}{g}\right) = O\left(n \cdot d \cdot \frac{e^g \cdot d^g}{w_0^g}\right) = O\left(\frac{n \cdot d \cdot (4e^2)^g \cdot d^g}{g^g \cdot M^{\frac{g}{g+1}}}\right) = O\left(\frac{n \cdot r \cdot d \cdot (4e^2)^g}{M^{\frac{g}{g+1}}}\right). \quad \square$$

Finally, note that the discussion of Case II above implicitly assumes $g \leq d$. Otherwise (or in fact, already if $d = O(g)$), then the conditions for Case II cannot hold at all. Indeed, $d = O(g)$ means $d \cdot r \leq O(g) \cdot \left(\frac{O(g)}{g}\right)^g \leq \delta^g$ for some constant δ. As such, $M = o(d \cdot r)$ implies $M < \delta^g$, i.e. $g = \Omega(\log M)$, which contradicts $g = o(\log M)$. So if $d = O(g)$, then Case II is in fact an empty regime: we are always in one of the remaining three cases. However, note that for realistic choices of the parameters, $g \leq d$ typically holds, and hence Case II is indeed non-empty.

3.5 Case III: when $w^* \leq g$

In Case III, the optimal w^* for square tiles would be smaller than g. However, many of the entries in $\mathbf{U}_1$ still require at least g values to generate. The best upper bounds here are mostly obtained simply

via the generic bounds in Lemma 3.1; this is also what we find in Theorem 1.3. Recall from the introduction that for $g = O(1)$, this is tight to the lower bound up to a constant factor. Note that even for $g = \omega(1)$, the bounds are also tight in some cases: for instance, if we have $d = O(g)$, then $\frac{n \cdot r \cdot d^2}{M}$ again matches the lower bound up to a constant factor. On the other hand, when $d = \Omega(g^2)$, one can show that the gap between the bounds in Theorem 1.3 is at least $g = \omega(1)$.

The regime of d between $O(g)$ and $\Omega(g^2)$ is more challenging. With some further work, we can actually extend the tight upper bounds to d being almost quadratic in g. The proof of this is more technical; it essentially adapts the upper bound idea from Case II with a choice of $w = g$, and some further adjustments. It also requires the extra assumption that M is polynomial in g.

Lemma 3.3. *In Case III, if we also assume that $d = O(g^{2-\delta_1})$ and $M = O(g^{\delta_2})$ for some constants $\delta_1, \delta_2 > 0$, then the optimal I/O complexity of Approximate Attention is upper bounded by $O(\frac{n \cdot r \cdot d \cdot g}{M})$.*

The lower bound in Theorem 1.3 uses the same tools (S-partitions) as in Case II, but requires a more complex proof. Intuitively, the terms that can be generated from less than g variables have a much larger role here. Due to this, we add an extra assumption here that $d \geq 5g$, which is indeed realistic if e.g. $g = O(1)$. With $d \geq 5g$, a technical lemma shows that at least a constant fraction of the columns in $\mathbf{U}_1$ use at least $\frac{g}{2}$ different variables. We can then restrict our analysis to this sub-matrix where any term requires loading $\frac{g}{2}$ values in $\mathbf{Q}$, and use a similar proof to Case II.

The formal proofs of Lemma 3.3 and the lower bound in Theorem 1.3 are available in Appendix B.

3.6 Case IV

Finally, for tiny cache sizes $M = O(g^2)$, we actually have another special case. Intuitively, here it becomes more beneficial to store the values of $\mathbf{U}_1$ than to recompute them; this provides tight bounds of $\Theta\left(\frac{n \cdot d \cdot r}{\sqrt{M}}\right)$, similarly to standard matrix multiplication. These proofs are discussed in Appendix C.

4 Discussion and Conclusion

In this work we studied the I/O complexity of Approximate Attention. We provided sharp upper and lower bounds for different parameter regimes, and showed that this approach can have significantly lower I/O cost than state-of-the-art methods like FlashAttention.

We note that our upper bounds, as well as the lower bound of Case I, are algorithm-agnostic: they hold for the approximate attention problem in general. In contrast to this, the lower bounds in Cases II-IV are inherently specific to the algorithm of Alman and Song.

Our proof techniques may also prove useful in future works for obtaining I/O bounds in other settings with special kinds of matrix multiplication. For instance, consider a sparse attention approach where the attention matrix is guaranteed to have at most $O(d)$ non-zero entries in each row: we can derive upper bounds for this in a very similar way to Lemma 3.1. As for the lower bound proofs for Cases II-IV, one natural candidate for generalization here is the recent method of Aliakbarpour et al [2]: this presents an extension of the Alman-Song algorithm, where matrices are decomposed into a dense part (handled via polynomial approximation) and a sparse part with larger entries. In any such decomposition, if the computations in the sparse part have I/O cost below $O(n \cdot d)$ due to their sparsity, then the I/O analysis reduces to our bounds. More generally, while our lower bound proofs are more specific to this concrete setting, their high-level ideas may still carry over to other algorithms for attention or matrix multiplication variants.

While the theoretical bounds are promising to improve the I/O cost of attention algorithms, it is well-known that the theoretically-fastest algorithms are not always the best-performing ones in practice. For example, the combinatorial nature of the parameter r (which originates from the analysis of [4]) can be prohibitively large for some practical parameter regimes. It is an interesting direction for future work to also investigate the efficiency and applicability of our algorithms in practice.

References

[1] Amol Aggarwal and Josh Alman. Optimal-Degree Polynomial Approximations for Exponentials and Gaussian Kernel Density Estimation. In *Computational Complexity Conference (CCC)*,

page 1, 2022.

[2] Maryam Aliakbarpour, Vladimir Braverman, Junze Yin, and Haochen Zhang. Support Basis: Fast Attention Beyond Bounded Entries. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 325–333, 2026.

[3] Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More Asymmetry Yields Faster Matrix Multiplication. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2005–2039. SIAM, 2025.

[4] Josh Alman and Zhao Song. Fast Attention Requires Bounded Entries. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 63117–63135, 2023.

[5] Josh Alman and Hantao Yu. Improving the Leading Constant of Matrix Multiplication. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1933–1971. SIAM, 2025.

[6] Toni Böhnlein, Pál András Papp, and Albert-Jan N. Yzelman. Red-Blue Pebbling with Multiple Processors: Time, Communication and Memory Trade-Offs. In *International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 109–126. Springer, 2025.

[7] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv*, 2019.

[8] Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J Colwell, and Adrian Weller. Rethinking Attention with Performers. In *International Conference on Learning Representations (ICLR)*, 2021.

[9] Tri Dao. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

[10] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlasAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 16344–16359, 2022.

[11] Giannis Daras, Nikita Kitaev, Augustus Odena, and Alexandros G Dimakis. SMYRF: Efficient Attention using Asymmetric Clustering. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 6476–6489, 2020.

[12] Erik D. Demaine, Andrea Lincoln, Quanquan C. Liu, Jayson Lynch, and Virginia Vassilevska Williams. Fine-grained I/O Complexity via Reductions: New Lower Bounds, Faster Algorithms, and a Time Hierarchy. In *Innovations in Theoretical Computer Science Conference (ITCS)*, pages 34–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018.

[13] Erik D. Demaine and Quanquan C. Liu. Red-blue pebble game: Complexity of computing the trade-off between cache size and memory transfers. In *Proceedings of Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 195–204, 2018.

[14] Jeff Erickson, Ivor Van Der Hoog, and Tillmann Miltzow. Smoothing the gap between NP and ER. *SIAM Journal on Computing*, 53(6):FOCS20–102, 2022.

[15] Insu Han, Rajesh Jayaram, Amin Karbasi, Vahab Mirrokni, David Woodruff, and Amir Zandieh. HyperAttention: Long-context Attention in Near-Linear Time. In *International Conference on Learning Representations (ICLR)*, 2024.

[16] Jia-Wei Hong and Hsiang-Tsung Kung. I/O complexity: The red-blue pebble game. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 326–333, 1981.

[17] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. In *International Conference on Machine Learning (ICML)*, pages 5156–5165. PMLR, 2020.

[18] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The Efficient Transformer. In *International Conference on Learning Representations (ICLR)*, 2020.

[19] Grzegorz Kwasniewski, Marko Kabić, Maciej Besta, Joost VandeVondele, Raffaele Solcà, and Torsten Hoefler. Red-Blue Pebbling Revisited: Near Optimal Parallel Matrix-Matrix Multiplication. In *Proceedings of International Conference on High-Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–22, 2019.

[20] Pál András Papp, Aleksandros Sobczyk, and Albert-Jan N Yzelman. The Impact of Partial Computations on the Red-Blue Pebble Game. In *Proceedings of Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 328–338, 2025.

[21] Pál András Papp and Roger Wattenhofer. On the Hardness of Red-Blue Pebble Games. In *Proceedings of Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 419–429, 2020.

[22] Barna Saha and Christopher Ye. I/O Complexity of Attention, or How Optimal is FlashAttention? In *International Conference on Machine Learning (ICML)*, pages 43024–43042, 2024.

[23] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pages 68658–68685, 2024.

[24] Aleksandros Sobczyk. I/O complexity and pebble games with partial computations. *Information Processing Letters*, page 106637, 2026.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

[26] Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas Singh. Nyströmformer: A Nyström-based Algorithm for Approximating Self-Attention. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 35, pages 14138–14148, 2021.

[27] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big Bird: Transformers for Longer Sequences. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pages 17283–17297, 2020.

A Proof of Key Lemma 3.2

A.1 Upper bound proof

The upper bound proof follows the idea outlined in Section 3.4. We split the output matrix into tiles of height $\frac{M}{4w}$ and width w. This already specifies a row strip of size $\frac{M}{4w} \times r$ in $\mathbf{U}_1$ and a column strip of size $r \times w$ in $\mathbf{H}$ that are needed to compute this output tile. We aggregate the output tile in $\binom{\frac{d \cdot g}{w}}{g}$ steps, splitting the row strip of $\mathbf{U}_1$ horizontally (and the column strip of $\mathbf{H}$ vertically) into aggregation tiles.

The aggregation tiles are obtained as follows. We split the column indices $\{1, \ldots, d\}$ of $\mathbf{Q}$ into groups of size $\frac{w}{g}$: let $G_1 = \{1, \ldots, \frac{w}{g}\}$, $G_2 = \{\frac{w}{g}+1, \ldots, 2 \cdot \frac{w}{g}\}$, …, $G_{\frac{dg}{w}} = \{d - \frac{w}{g}+1, \ldots, d\}$. Each aggregation tile is formed by selecting g distinct groups, i.e. the number of aggregation tiles is $\binom{\frac{dg}{w}}{g}$. In each aggregation tile, we consider the union of the g groups combined, and consider the resulting w distinct column indices; specifically, for each of the corresponding $\frac{M}{4w}$ rows, we load the values from these w columns of $\mathbf{Q}$. This corresponds to $\frac{1}{4}M$ load operations for each aggregation tile.

Note that some of the terms in the polynomial require an entry from g distinct groups; these columns of $\mathbf{U}_1$ can only be generated on the specific tile that combines these g groups. Since any term contains

at most g variables, each entry of $\mathbf{U}_1$ can be generated in at least one tile. However, there will also be numerous terms that can be generated in multiple different tiles: e.g. a term that only uses variable indices in G_1 and G_2 can be assigned to any of the tiles that combine G_1, G_2 and a selection of $(g-2)$ arbitrary further groups. For simplicity, we will always assign each term to the first tile that can generate it. That is, we iterate through the group subsets via a lexicographic order of their sorted indices, and we assign each term to the first tile that can generate it in this order. Note that for any term, it is straightforward to determine the tile that generates it: we consider the groups that contain at least one variable of this term, and if the number of groups is smaller than g, we use the smallest non-included group indices for the rest of the groups. As such, for any tile and term, it is easy to decide whether the term is assigned to this given tile or not. This means that for any given tile, we can iterate over all the $\tau(w)$ terms that can be generated from the corresponding w inputs, and generate only the terms that belong to this specific tile, avoiding duplications. With this method, the width of each tile in $\mathbf{U}_1$ will be at most $\tau(w)$ (the maximum number of terms we can generate from w variables), and at least $(\frac{w}{g})^g$ (the terms using g distinct groups that can only be obtained on this tile).

We point out that in Case II, the aggregation tiles we form are always taller than wider. Indeed the height of the tiles is $\frac{M}{4w}$, and their width is at most $\tau(w)$. We have $\frac{M}{4w} \geq \tau(w)$ due to $w \cdot \tau(w) \leq \frac{1}{4}M$. This is crucial to ensure that the corresponding tiles formed in $\mathbf{H}$ indeed fit into fast memory: since their height is at most $\tau(w) \leq \frac{M}{4w}$, and their width is w, they contain at most $\frac{1}{4}M$ entries.

Altogether, each aggregation tile requires at most $\frac{1}{4}M$ loaded inputs from both $\mathbf{U}_1$ and $\mathbf{H}$, which are always kept in fast memory. The results are aggregated in a tile of size $\frac{1}{4}M$ in the output matrix, which are written to slow memory in the end. As such, each sub-computation fits into the working memory of M, and has I/O cost of $O(M)$. The number of tiles in the output matrix is $\frac{n}{\frac{M}{4w}} \cdot \frac{d}{w}$, and each are aggregated in $\binom{\frac{dg}{w}}{g}$ steps. This results in a total I/O cost of

$$\frac{4nw}{M} \cdot \frac{d}{w} \cdot \binom{\frac{dg}{w}}{g} \cdot M = O\left(n \cdot d \cdot \binom{\frac{dg}{w}}{g}\right).$$

We note that in the red-blue pebbling model, generating the entries in $\mathbf{U}_1$ requires no further working memory; the process of iterating through the entries is incorporated into the pebbling strategy. However, in a practical implementation, this requires a few further fast memory entries, e.g. index variables to iterate through the $\tau(w)$ possible combinations.

We also note that when aggregating the results in a tile of size $\frac{1}{4}M$ above, this actually incurs an I/O cost of $\frac{1}{2}M$, i.e. two I/O steps for each entry of the matrix (in the appropriate red-blue pebbling variant; see Appendix A.3 for a discussion). Indeed, we first need to load the current aggregated value for the entry in the output matrix, add the newly computed terms to this value, and then save the new (partial) aggregated value to slow memory again. However, this is simply a clarification of a technical detail; since this cost is still in $O(M)$, it has no effect on the analysis of I/O costs above.

Extending the algorithm to the entire computation $\mathbf{U}_1(\mathbf{U}_2^\top \mathbf{V})$ is straightforward. We simply execute the two multiplications separately, considering $\mathbf{H} = \mathbf{U}_2^\top \mathbf{V}$ as an output in the first and as an input in the second. In the first computation, we use the same tiling strategy as discussed above, just with the role of the matrices exchanged. That is, we form column strips of width $\frac{M}{4w}$ in $\mathbf{U}_2^\top$ and the corresponding row strips of height $\frac{M}{4w}$ in $\mathbf{V}$. We split each of these strips to $\binom{\frac{dg}{w}}{g}$ tiles (of differing height) in $\mathbf{U}_2^\top$. This results in tiles of width w and height at most $\tau(w)$ in $\mathbf{H}$. Each such tile will be aggregated in $\frac{n}{\frac{M}{4w}}$ steps from the different column strips in $\mathbf{U}_2^\top$ and row strips in $\mathbf{V}$. With $\binom{\frac{dg}{w}}{g} \cdot \frac{d}{w}$ tiles in $\mathbf{H}$, the total I/O cost is the same as in the second matrix multiplication.

We point out that in almost all cases, the upper bound for Theorem 1.2 (obtained via Key Lemma 3.2) is tighter than the generic upper bound from Lemma 3.1. In particular, when $M = O(d^2)$ and hence the generic upper bound is $O(\frac{n \cdot r \cdot d}{\sqrt{M}})$, then our upper bound in Theorem 1.2 is smaller when $M^{\frac{1}{2}-\frac{1}{g+1}} \geq (4e^2)^g$ holds. For any $g \geq 2$, the left side here is lower bounded by $M^{\frac{1}{6}}$, and then the claim indeed follows from $g = o(\log M)$. On the other hand, when $M = \omega(d^2)$, and hence the generic upper bound is $O(\frac{n \cdot r \cdot d^2}{M})$, then our upper bound is smaller when $d \geq M^{\frac{1}{g+1}} \cdot (4e^2)^g$. This also holds if we have any constant $\delta > 0$ such that $M \leq d^\delta$. Indeed, then $(4e^2)^g$ can be

upper bounded by $M^{\frac{\delta}{2}}$ (because $g = o(\log M)$) and $M^{\frac{1}{g+1}}$ can also be upper bounded by $M^{\frac{\delta}{2}}$ asymptotically (assuming that $g = \omega(1)$; otherwise, the upper and lower bounds are tight anyway). The generic upper bound may only be superior for very specific parameter combinations, e.g. if we have $M = \left(\frac{d}{g}\right)^{g+1}$.

A.2 Lower bound proof

The lower bound proof corresponds to the hypothetical case above where each set of w entries from $\mathbf{q}$ generates $\tau(w)$ distinct entries in $\mathbf{u}$, and hence it is likely somewhat lower than the actual optimum.

When proving lower bounds for I/O cost, one of the most widely used tools are so-called S-partitions. Defined by Hong and Kung in their original paper on red-blue pebbling and I/O complexity [16], an S-partition is a way to partition the node of the computational graph such that, intuitively, each class can be computed with only $2S$ I/O steps.

Definition A.1. *An* S-partition *of a computational DAG (for some integer parameter* S*) is a disjoint partitioning* D_1*, ...,* D_k *of the vertices of the DAG such that*

- *the classes* D_1*, ...,* D_k *form a topological order of the computation, i.e. there are no edges from* D_j *to* D_i *with* $i < j$*;*
- *for each class* D_i*, there is a so-called* dominator set *of size* S *in the DAG, i.e. a set of nodes such that every path from a source node to a node in* D_i *contains a node in this set;*
- *for each class* D_i*, the so-called* minimum set *of* D_i*, i.e. the set of nodes in* D_i *without a child in* D_i*, has size at most* S*.*

For more details on S-partitions, we refer the reader to [16]. The crucial property of S-partitions is that they directly allow to lower bound the optimal I/O cost when using $S = 2M$.

Lemma A.1 (From Hong & Kung [16])**.** *If* MIN_{2M} *denotes the minimal number of classes in any* $2M$*-partition of a computational DAG, then the optimal I/O cost of this computation is at least* $M \cdot (\text{MIN}_{2M} - 1)$.

In the lower bound proof, we analyze the number of classes that each $2M$-partition must have in our computation. Through the lemma above, this directly yields the lower bound in Key Lemma 3.2.

Proof of Key Lemma 3.2, lower bound. Consider first the computational DAG restricted to the multiplication $\mathbf{U}_1\mathbf{H}$, i.e. where the entries of $\mathbf{H}$ are all source nodes. Let us refer to the nodes that correspond to (the output of) multiplying an entry of $\mathbf{U}_1$ and an entry of $\mathbf{H}$ as *internal nodes*. The DAG altogether has $n \cdot r \cdot d$ internal nodes. Similarly to previous I/O lower bounds on matrix multiplication variants, the main idea of the proof is to upper bound the number of internal nodes that can be contained in each partition, thus obtaining a lower bound on MIN_{2M}.

Let D_i be any partition in a $2M$-partition. Consider the minimum set of D_i and a dominator set of size at most $2M$ for D_i. This is altogether at most $4M$ nodes; even if they are all internal, this gives at most $4M$ internal nodes. Let Γ be the set of all other internal nodes contained in D_i; let us call these *uncovered* internal nodes. For every uncovered node $v \in \Gamma$, we must have that (i) on each directed path from a source node to v, we have a node in the dominator set, and (ii) at least one successors of v is in the minimum set. Note that each internal node has exactly one parent in $\mathbf{H}$ that is a source node, so for uncovered nodes, these must all be in the dominator set. An uncovered node $v \in \Gamma$ also has a parent in $\mathbf{U}_1$; however, here there are two options, either the parent in $\mathbf{U}_1$ is in the dominator set, or all the parents of the parent (located in $\mathbf{Q}$) are in the dominator set. As for the successor of v in the minimum set, this may be the sink node in the output matrix that corresponds to v, or any node in the summation tree from v to the sink.

For any $i \in [n]$, let ${w_i}^{(\mathbf{Q})}$ and ${w_i}^{(\mathbf{U}_1)}$, respectively, denote the number of entries in the i^{th} row of $\mathbf{Q}$ and the i^{th} row of $\mathbf{U}_1$ that are in the dominator set. Let $w_i = {w_i}^{(\mathbf{Q})} + {w_i}^{(\mathbf{U}_1)}$. Let us consider the number of nodes in the i^{th} row of $\mathbf{U}_1$ that can have a child in Γ. These nodes must either be in the dominator set, or have all their parents in the dominator set, so their number is at most $\tau({w_i}^{(\mathbf{Q})}) + {w_i}^{(\mathbf{U}_1)}$. With $w_i = {w_i}^{(\mathbf{Q})} + {w_i}^{(\mathbf{U}_1)}$, this is at most $\tau(w_i)$.

Let us sort $\mathbf{U}_1$ into two sub-matrices: let $\mathbf{U}_1^{(a)}$ contain those rows where $w_i \geq w$, and $\mathbf{U}_1^{(b)}$ contain those rows where $w_i < w$. The matrix multiplication can similarly be split into two sub-multiplications $\mathbf{U}_1^{(a)}\mathbf{H}$ and $\mathbf{U}_1^{(b)}\mathbf{H}$. In $\mathbf{U}_1^{(a)}$, each row has at least w nodes in the dominator set, so the number of rows is at most $\frac{2M}{w}$. On the other hand, the number of nodes of $\mathbf{H}$ in the dominator set is also at most $2M$. Each row in $\mathbf{U}_1^{(a)}$ has only one value that is multiplied with any of the $2M$ nodes in $\mathbf{H}$, hence the number of uncovered internal nodes in $\mathbf{U}_1^{(a)}\mathbf{H}$ is at most $\frac{2M}{w} \cdot 2M = \frac{4M^2}{w}$.

On the other hand, in $\mathbf{U}_1^{(b)}$, each row can have at most $\tau(w) \leq \frac{M}{4w}$ nodes that have a child in Γ. Recall that each node in Γ has a successor in the minimum set; this means that there are at most $2M$ sink nodes in the output matrix such that each node in Γ is a predecessor of one of these sink nodes. For such a sink node v_0, we can have at most $\frac{M}{4w}$ nodes in Γ that are predecessors of v_0, because at most $\frac{M}{4w}$ nodes of $\mathbf{U}_1$ can have a child in Γ in the given row. This limits the number of uncovered internal nodes in $\mathbf{U}_1^{(b)}\mathbf{H}$ to $\frac{M}{4w} \cdot 2M = \frac{M^2}{2w}$.

Altogether, the number of nodes in Γ is at most $4M + \frac{4M^2}{w} + \frac{M^2}{2w} = O(\frac{M^2}{w})$. With $n \cdot r \cdot d$ internal nodes, this means that the number of partitions is at least $\Omega(\frac{n \cdot r \cdot d \cdot w}{M^2})$. Using Lemma A.1, we get that the optimal I/O cost is at least $\Omega(2 \cdot \frac{n \cdot r \cdot d \cdot w}{M^2} - 1) \cdot M = \Omega(\frac{n \cdot r \cdot d \cdot w}{M})$. This finishes the proof of the lower bound.

In order to extend the proof from the multiplication $\mathbf{U}_1\mathbf{H}$ to the entire computation $\mathbf{U}_1(\mathbf{U}_2^\top \mathbf{V})$, we need to consider the fact that the nodes in $\mathbf{H} = \mathbf{U}_2^\top \mathbf{V}$ are in fact not sources. That is, instead of having an entry of $\mathbf{H}$ in the dominator set, we could include some of its predecessors. However, each node in $\mathbf{H}$ has n parents in the same column of $\mathbf{V}$, which would all need to be in the dominator set in this case. As such, if a dominator set in our previous proof contained $x \in \{0, ..., r\}$ entries from a specific column of $\mathbf{H}$, we could only replace these by n nodes in $\mathbf{V}$ (and further nodes in $\mathbf{U}_2$ or $\mathbf{K}$, but these can be ignored now). Intuitively, with $n > r$, this only increases the size of the dominator set. As such, for the whole computation $\mathbf{U}_1(\mathbf{U}_2^\top \mathbf{V})$, it still holds that there can be at most $2M$ nodes in $\mathbf{H}$ with a child that belongs to Γ, which allows us to bound the number of uncovered internal nodes in $\mathbf{U}_1^{(b)}\mathbf{H}$ exactly as before. □

A.3 Discussion on red-blue pebbling variants

We note that the standard version of the red-blue pebble game raises some modeling questions regarding operators with many inputs, such as the summation part in the matrix multiplication. If the summation is modeled through a single node, then this requires all inputs in fast memory at the same time, which means that there is no viable pebbling strategy at all if $M \leq r$.

Previous works have often avoided the discussion of this issue by forming a binary 'summation tree' in the computational DAG, and assuming that aggregation happens this way [22]. This is in fact a strong restriction on the execution strategies considered. However, for heavily symmetric operations like matrix multiplication, this had no effect on the optimal I/O complexity: since the optimal strategy was to form tiles of specific sizes, the tiles can naturally consist of neighboring rows/columns in the matrices, and hence the summation within a tile essentially corresponds to a sub-tree in the summation tree of the final aggregated value.

The same summation tree approach becomes somewhat more problematic for our Approximate Attention problem. Our results indicate that for an I/O-efficient execution, we need to group together specific columns of the matrix $\mathbf{U}_1$; hence it becomes critical that the corresponding group of nodes also form a sub-tree in the summation tree of the given entry of the output matrix. As such, this requires a model that inherently entangles the DAG representation of the computation with the optimal I/O strategy to execute it.

In general, a more appropriate approach to resolve the question of summation trees is to consider a generalization of the red-blue pebble game with partial computations [20, 24]. This generalized model allows to keep the clean representation of the aggregations as a single node with r incoming edges (as in Figure 1), requiring no summation trees at all. The aggregation process is then modeled by the pebble game itself, which allows to store partially computed results in fast or slow memory. This extended model is a much more accurate description of how the computation is executed in practice. Our algorithm descriptions in the upper bounds are also easiest to interpret in this model.

On the other hand, deriving I/O lower bounds for this generalized pebble game also requires a small adjustment to the S-partitioning concept, using so-called S-edge-partitions [20]. While our lower bound proofs are presented in terms of the standard red-blue pebble game and S-partitions, they also carry over to this generalized model and S-edge-partitions by simply shifting the focus from the internal nodes to their (single) outgoing edges, similarly to the proofs adaptations for classic computations [20].

B Proof of Theorem 1.3

B.1 Upper bound discussion

The upper bound in Theorem 1.3 simply re-states the generic upper bounds from Lemma 3.1. Note that once again when $g = O(1)$, the bounds are tight up to a constant factor. For instance, the factor of difference between $\frac{n \cdot r \cdot d}{\sqrt{M}}$ and $\frac{n \cdot r \cdot d \cdot g}{M}$ is $\frac{\sqrt{M}}{g}$. Due to $g = \Omega(\log M)$, we have $M = 2^{O(g)}$; for $g = O(1)$, this implies $M = O(1)$, and thus $\frac{\sqrt{M}}{g} = O(1)$.

We point out that besides $g = O(1)$, the bounds are also tight for some other parameter configurations. That is, let us assume $g = \omega(1)$. In this case, for instance when we have $d = O(g)$, then $\frac{n \cdot r \cdot d^2}{M}$ is again only a constant factor away from the lower bound. On the other hand, when $d = \Omega(g^2)$, the gap between the upper and lower bound is $\frac{\min(d, \sqrt{M})}{g}$, which is at least g (since $M = \Omega(g^2)$). As such, this case exhibits a larger gap than a constant factor.

At this point, it is a natural question how the upper bounds behave when d is between g and g^2 (with $g = \omega(1)$). As a further contribution, we present another algorithm that is very similar to that of Key Lemma 3.2, but adapted to Case III, in order to answer this question. The upper bound obtained by this algorithm will be weaker than the generic bounds of Lemma 3.1 when $d = \Omega(g^2)$. However, for the case when there are constants $\delta_1, \delta_2 > 0$ such that $d = O(g^{2-\delta_1})$ and $M = O(g^{\delta_2})$, this algorithm will once again provide a tight upper bound of $O(\frac{n \cdot r \cdot d \cdot g}{M})$; this is formally stated in Lemma 3.3. This result extends the tight upper bound from $d = O(g)$ to d almost quadratic in g, assuming that M is at most polynomial in g.

B.2 Detour: proof of Lemma 3.3

More specifically, our algorithm for Lemma 3.3 proves an upper bound of

$$O\left(n \cdot d \cdot \left(\binom{d}{g} + \frac{4 \cdot r \cdot g}{M}\right)\right) \tag{3}$$

when $d = O(g^{2-\delta_1})$ and $M = O(g^{\delta_2})$ for some constant $\delta > 0$.

We first show that in our parameter regime, we have $\binom{d}{g} \leq \frac{4 \cdot r \cdot g}{M}$; this implies that the formula in Equation (3) can be upper bounded by $\frac{8 \cdot n \cdot r \cdot d \cdot g}{M}$, which is indeed a constant factor away from the lower bound of Theorem 1.3.

The claim we need to prove is equivalent to

$$\frac{M}{4g} \leq \frac{r}{\binom{d}{g}} = \frac{\binom{d+g}{g}}{\binom{d}{g}} = \frac{(d+g) \cdot \ldots \cdot (d+1)}{d \cdot \ldots \cdot (d-g+1)}.$$

The expression on the right-hand side can be lower bounded by $(\frac{d+g}{d})^g$. With $d = O(g^{2-\delta_1})$, this can be further lower bounded by $\left(\frac{g^{2-\delta_1'}+g}{g^{2-\delta_1'}}\right)^g$ for some constant δ_1' such that $\delta_1 > \delta_1' > 0$. This is then

$$\left(\frac{g^{1-\delta_1'}+1}{g^{1-\delta_1'}}\right)^g = \left(1 + \frac{1}{g^{1-\delta_1'}}\right)^g = \left(\left(1 + \frac{1}{g^{1-\delta_1'}}\right)^{(g^{1-\delta_1'})}\right)^{(g^{\delta_1'})}.$$

The outer parenthesis goes to e as g goes to infinity, hence from some point we can lower bound it by $\frac{e}{2}$. We also have $M = O(g^{\delta_2}) \leq g^{\delta_2'}$ for some $\delta_2' > \delta_2$. Hence we only need to show

$g^{\delta_2'} \leq 4 \cdot g \cdot \left(\frac{e}{2}\right)^{(g^{\delta_1'})}$; this indeed holds asymptotically for any constants δ_1', δ_2', since the left side is polynomial, while the right side is exponential in g.

It remains to show the I/O strategy that proves the upper bound in Equation (3). This is very similar to the upper bound proof of Key Lemma 3.2, with a specific choice of $w = g$. That is, we split the output matrix into tiles of height $\frac{M}{4g}$ and width g. The row strip of size $\frac{M}{4g} \times r$ in $\mathbf{U}_1$ is split into $\binom{d}{g}$ aggregation tiles horizontally, and the column strips of width g in $\mathbf{H}$ are similarly split vertically.

For the aggregation tiles, now each subset of $\{1, \ldots, d\}$ of size g will form a separate tile; this corresponds to having $|G_1| = \ldots = |G_d| = 1$ in the key lemma proof. In each aggregation tile, we load the corresponding g values; this gives one term that can only be generated on this tile, and other terms that can be generated in multiple tiles. We once again assign each of the latter terms to the first possible tile in the lexicographic order. Similarly to the analysis before, this approach results in $\frac{n}{\frac{M}{4g}} \cdot \frac{d}{g}$ tiles in the output matrix.

The key difference to Key Lemma 3.2 here is that the width of the tiles formed in $\mathbf{U}_1$ might be larger than their height: that is, we may have $\tau(g) > \frac{M}{4g}$. In this case, the corresponding tile in $\mathbf{H}$, which has size $\tau(g) \times g$, might not fit into fast memory. Due to this, we split each of these tiles further into sub-tiles of height at most $\frac{M}{4g}$; with this, the corresponding tile in $\mathbf{H}$ has size $\frac{M}{4g} \cdot g = \frac{1}{4}M$. Executing this split only requires us to memorize the current position in the aggregation tile in $\mathbf{U}_1$. For each sub-tile, the same $\frac{M}{4g} \times g$ values can be kept from $\mathbf{Q}$, the tile aggregated in the output can also remain in fast memory. However, from $\mathbf{H}$, we need to load up to $\frac{1}{4}M$ new values for each sub-tile. Let the width of the tiles in $\mathbf{U}_1$ be $t_1, t_2, \ldots$; note that their sum is r. Then the number of times we need to start a new sub-tile within a tile is at most $\frac{t_1}{\frac{M}{4g}} + \frac{t_1}{\frac{M}{4g}} + \ldots = \frac{r}{\frac{M}{4g}} = \frac{4 \cdot r \cdot g}{M}$. With $\binom{d}{g}$ tiles, this means that the number of sub-tiles is at most $\binom{d}{g} + \frac{4 \cdot r \cdot g}{M}$ altogether. This results in a total I/O cost of

$$\frac{n}{\frac{M}{4g}} \cdot \frac{d}{g} \cdot \left(\binom{d}{g} + \frac{4 \cdot r \cdot g}{M}\right) \cdot M \,,$$

which is equivalent to Equation (3).

The strategy extends to the multiplication $\mathbf{U}_2^\top \mathbf{V}$ exactly as in the proof of Key Lemma 3.2: we split $\mathbf{U}_2^\top$ to column strips of width $\frac{M}{4g}$, split each of these to $\binom{d}{g}$ parts vertically, and form $\binom{d}{g} \cdot \frac{d}{g}$ tiles in the output $\mathbf{H}$, each of which is aggregated over $\frac{n}{\frac{M}{4g}}$ steps. Similarly to the second multiplication, we might have tiles in $\mathbf{U}_2^\top$ that are taller than $\frac{M}{4g}$; these are then split vertically into sub-tiles of height $\frac{M}{4g}$ at most. For all of the sub-tiles, the tile of size $g \times \frac{M}{4g}$ in $K^\top$ and the tile of size $\frac{M}{4g} \times g$ in $\mathbf{V}$ can remain the same, whereas in $\mathbf{H}$, we save different sub-tiles each time.

B.3 Lower bound proof

The lower bound proof of Theorem 1.3 is similar to that in Key Lemma 3.2. However, here we only focus on the columns of $\mathbf{U}_1$ that have at least $\frac{g}{2}$ distinct parents in $\mathbf{Q}$. One can show that if $d \geq 5g$, this indeed amounts to a constant fraction of the columns of $\mathbf{U}_1$. This follows from the lemma below, which can be understood as the reverse direction: the number of terms generated from at most $(\frac{g}{2} - 1)$ variables is at most $\delta \cdot r$.

Lemma B.1. *For $d \geq 5g$, there exists a constant $0 < \delta < 1$*

$$\binom{d}{\frac{g}{2} - 1} \cdot \tau\left(\frac{g}{2} - 1\right) \leq \delta \cdot r \,.$$

Since this is only a technical lemma that offers no further insight into the proof, we defer its proof to Appendix D.

In our proof, we only consider a subset of the matrix multiplication: the at least $\delta \cdot r$ columns in $\mathbf{U}_1$ that satisfy this condition, and the corresponding at least $\delta \cdot r$ rows in $\mathbf{H}$. We assume that all other I/O operations are free; the lower bound derived this way clearly carries over to the original matrix multiplication.

The proof begins similarly to Key Lemma 3.2. Let D_i be a partition in a $2M$-partition of this computation. Consider its minimum set and a dominator set of size at most $2M$. Let Γ be the set of all the uncovered internal nodes in D_i that are not among these $4M$ nodes. All uncovered nodes $v \in \Gamma$ must have their parent in $\mathbf{H}$ in the dominator set, and also either their parent in $\mathbf{U}_1$ or all the parents of this parent in the dominator set.

For any $i \in [n]$, let ${w_i}^{(\mathbf{Q})}$ and ${w_i}^{(\mathbf{U}_1)}$ again be, respectively, the number of entries in the i^{th} row of $\mathbf{Q}$ and the i^{th} row of $\mathbf{U}_1$ that are in the dominator set. Let $w_i = {w_i}^{(\mathbf{Q})} + {w_i}^{(\mathbf{U}_1)}$. Consider two sub-matrices of $\mathbf{U}_1$: let $\mathbf{U}_1^{(a)}$ contain the rows with $w_i \geq \frac{g}{2}$, and $\mathbf{U}_1^{(b)}$ contain the rows with $w_i < \frac{g}{2}$. This splits the matrix multiplication into two sub-multiplications $\mathbf{U}_1^{(a)}\mathbf{H}$ and $\mathbf{U}_1^{(b)}\mathbf{H}$. In $\mathbf{U}_1^{(a)}$, each row has at least $\frac{g}{2}$ nodes in the dominator set, so the number of rows is at most $\frac{2M}{\frac{g}{2}} = \frac{4M}{g}$. The number of nodes of $\mathbf{H}$ in the dominator set is at most $2M$. Each row in $\mathbf{U}_1^{(a)}$ has only one value multiplied with any entry in $\mathbf{H}$, so the number of uncovered internal nodes in $\mathbf{U}_1^{(a)}\mathbf{H}$ is at most $\frac{4M}{g} \cdot 2M = \frac{8M^2}{g}$.

In $\mathbf{U}_1^{(b)}$, none of the nodes in (our restricted) $\mathbf{U}_1$ can have all their parents in the dominator set; hence if a node in Γ has its parent in $\mathbf{U}_1^{(b)}$, then this parent has to be in the dominator set. Thus for simplicity, we can assume that ${w_i}^{(\mathbf{Q})} = 0$, i.e. none of the source nodes are in the dominator set, since these nodes could be removed from the set anyway. With ${w_i}^{(\mathbf{U}_1)} < \frac{g}{2}$, each of the rows in $\mathbf{U}_1^{(b)}$ can have at most $\frac{g}{2}$ nodes which have a child in Γ. Similarly to before, each node in Γ must also have a successor in the minimum set, so there are at most $2M$ sink nodes in the output matrix with a predecessor in Γ. Each sink node has at most $\frac{g}{2}$ predecessors in Γ, since number of nodes in $\mathbf{U}_1^{(b)}$ with a child in Γ is at most $\frac{g}{2}$ in the given row. This means that the number of uncovered internal nodes in $\mathbf{U}_1^{(b)}\mathbf{H}$ is at most $\frac{g}{2} \cdot 2M = g \cdot M$.

In Case III, we have $M = \omega(g^2)$. This means that $g \cdot M$ is also upper bounded by $O(\frac{M^2}{g})$, so the total number of nodes in Γ is at most $4M + \frac{8M^2}{g} + g \cdot M = O(\frac{M^2}{g})$. Having $n \cdot r \cdot d$ internal nodes, the number of partitions is at least $\Omega(\frac{n \cdot r \cdot d \cdot g}{M^2})$. Using Lemma A.1, the optimal I/O cost in this case is at least $\Omega(\frac{n \cdot r \cdot d \cdot g}{M})$.

The proof can be extended to the entire computation $\mathbf{U}_1(\mathbf{U}_2^\top \mathbf{V})$ exactly as in Key Lemma 3.2.

C Proof of Theorem 1.4

The upper bound in Theorem 1.4 follows easily from the generic upper bound in Lemma 3.1.

For the lower bound, the proof starts identically to that of Theorem 1.3. We only consider the columns of $\mathbf{U}_1$ with at least $\frac{g}{2}$ distinct parents in $\mathbf{U}_1$. This once again amounts to a constant fraction of the columns of $\mathbf{U}_1$.

For any row $i \in [n]$, let w_i, ${w_i}^{(\mathbf{Q})}$ and ${w_i}^{(\mathbf{U}_1)}$ be as before. We split the matrix again into $\mathbf{U}_1^{(a)}$ for rows with $w_i \geq \frac{g}{2}$, and $\mathbf{U}_1^{(b)}$ for rows with $w_i < \frac{g}{2}$. In the sub-multiplication $\mathbf{U}_1^{(a)}\mathbf{H}$ the number of uncovered internal nodes in $\mathbf{U}_1^{(a)}\mathbf{H}$ is at most $\frac{8M^2}{g}$ as before.

Otherwise, the proof is analogously to the lower bound for standard matrix multiplication [16], essentially repeating the same arguments with a split at $\sqrt{M}$ instead of $\frac{g}{2}$. Recall that for each row in $\mathbf{U}_1^{(b)}$, we can assume ${w_i}^{(\mathbf{Q})} = 0$. Let us further divide the matrix $\mathbf{U}_1^{(b)}$, with $\mathbf{U}_1^{(b_1)}$ containing the rows of $\mathbf{U}_1^{(b)}$ where ${w_i}^{(\mathbf{U}_1)} \geq \sqrt{M}$, and $\mathbf{U}_1^{(b_2)}$ containing the rows of $\mathbf{U}_1^{(b)}$ where ${w_i}^{(\mathbf{U}_1)} < \sqrt{M}$. In $\mathbf{U}_1^{(b_1)}$, the number of rows is at most $\frac{2M}{\sqrt{M}} = 2\sqrt{M}$. Each of these rows has at most one value multiplied with any of the (at most $2M$) nodes of $\mathbf{H}$ in the dominator set, so the number of nodes of Γ in $\mathbf{U}_1^{(b_1)}\mathbf{H}$ is at most $2M \cdot 2\sqrt{M} = 4M \cdot \sqrt{M}$. In $\mathbf{U}_1^{(b_2)}$, we again consider the at most $2M$ sink nodes in the output matrix that have a predecessor in Γ. Each of these can have at most $\sqrt{M}$

predecessors in Γ, since the number of nodes in Γ in any row is at most $\sqrt{M}$. This means that Γ has at most $2M \cdot \sqrt{M}$ nodes in $\mathbf{U}_1^{(b_2)}$.

Altogether, the number of nodes of Γ in $\mathbf{U}_1^{(b)}\mathbf{H}$ in this case is at most $6M \cdot \sqrt{M}$. Together with the nodes in $\mathbf{U}_1^{(a)}\mathbf{H}$ and the internal nodes directly in the dominator/minimum sets, this is at most $4M + \frac{8M^2}{g} + 6M \cdot \sqrt{M}$ nodes. Since Case IV has $M = O(g^2)$, we also have $\frac{M^2}{g} = O(M \cdot \sqrt{M})$. As such, this expression is altogether in $O(M \cdot \sqrt{M})$. With $n \cdot r \cdot d$ internal nodes, the number of partitions is $\Omega(\frac{n \cdot r \cdot d}{M \cdot \sqrt{M}})$, and the optimal I/O cost according to Lemma A.1 is $\Omega(\frac{n \cdot r \cdot d}{\sqrt{M}})$.

D Technical lemmas

D.1 Using attention to compute the exponential function

Assume that attention can be computed exactly in T steps. To compute the exponential $\exp(x)$ for any x, simply set $\mathbf{Q} = 1$, $\mathbf{K}^\top = (x \quad 2x)$, $\mathbf{V} = (1; 0)$, then compute $\mathbf{A} = \mathrm{Att}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \frac{\exp(x)}{\exp(x)+\exp(2x)}$, and finally return $z = 1/\mathbf{A} - 1 = \exp(x)$.

D.2 Closed form for $\tau(w)$

For completeness, we establish the closed-form expression on $\tau(w)$.

Lemma D.1. *For any integer $w \geq 1$, we have*

$$\sum_{l=0}^{g} \binom{w+l-1}{l} = \binom{w+g}{g}.$$

Proof. We can show this by an induction on g. The claim clearly holds for small values. For $g = 0$, we have $\binom{w-1}{0} = 1$ on the left-hand side and $\binom{w}{0} = 0$ on the right-hand side. For $g = 1$, we have $1 + \binom{w}{1} = w + 1$ on the left-hand side and $\binom{w+1}{1} = w + 1$ on the right-hand side.

Now consider $g \geq 2$, and assume the claim holds for $(g-1)$. This means that we only need to show

$$\binom{w+g-1}{g} = \binom{w+g}{g} - \binom{w+(g-1)}{(g-1)}.$$

Indeed, the right-hand side is identical to

$$\left(\frac{(w+g)\cdot(w+g-1)\cdot\ldots\cdot(w+1)}{g!} - \frac{(w+g-1)\cdot(w+g-2)\cdot\ldots\cdot(w+1)}{(g-1)!}\right) =$$

$$(w+g)\cdot\frac{(w+g-1)\cdot\ldots\cdot(w+1)}{g!} - g\cdot\frac{(w+g-1)\cdot\ldots\cdot(w+1)}{g!} =$$

$$\frac{(w+g-1)\cdot\ldots\cdot(w+1)\cdot w}{g!} = \binom{w+g-1}{g}.$$

□

D.3 Proof of Lemma B.1

Below we prove Lemma B.1, i.e. that for $d \geq 5g$, we have a constant $0 < \delta < 1$ with

$$\binom{d}{\frac{g}{2}-1} \cdot \tau\left(\frac{g}{2}-1\right) \leq \delta \cdot r.$$

Proof. For simplicity, we instead prove the stronger statement

$$\binom{d}{\frac{g}{2}} \cdot \tau\left(\frac{g}{2}-1\right) \leq \delta \cdot r.$$

Using the definition of r and τ, this is equivalent to

$$\binom{d}{\frac{g}{2}} \cdot \binom{\frac{3}{2}g-1}{\frac{g}{2}-1} \leq \delta \cdot \binom{d+g}{g}.$$

This can be further expanded to

$$\frac{d!}{(\frac{g}{2})! \cdot (d-\frac{g}{2})!} \cdot \frac{(\frac{3}{2}g-1)!}{(\frac{g}{2}-1)! \cdot g!} \leq \delta \cdot \frac{(d+g)!}{d! \cdot g!}$$

and then

$$d \cdot \ldots \cdot (d-\frac{g}{2}+1) \cdot \frac{(\frac{3}{2}g-1) \cdot \ldots \frac{g}{2}}{(\frac{g}{2})!} \leq \delta \cdot (d+g) \cdot \ldots \cdot (d+1).$$

We can upper bound $(\frac{3}{2}g-1) \cdot \ldots \frac{g}{2}$ by $\frac{1}{2} \cdot g^g$. We can lower bound $(\frac{g}{2})!$ by $(\frac{g}{5})^{g/2}$. We also have

$$\frac{(d+g) \cdot \ldots \cdot (d+1)}{d \cdot \ldots \cdot (d-\frac{g}{2}+1)} \geq d^{\frac{g}{2}}.$$

Hence it is enough to prove that

$$\frac{1}{2} \cdot \frac{g^g}{(\frac{g}{5})^{g/2}} = \frac{1}{2} \cdot 5^{\frac{g}{2}} \cdot g^{\frac{g}{2}} \leq \delta \cdot d^{\frac{g}{2}}.$$

This is equivalent to

$$\frac{d}{5g} \geq \left(\frac{1}{4 \cdot \delta^2}\right)^{\frac{1}{g}}.$$

By selecting a constant δ close to 1, we have $\left(\frac{1}{4 \cdot \delta^2}\right) \leq \frac{1}{3}$. The left side is then upper bounded by 1 for any $g \geq 1$, and hence it is sufficient to have $d \geq 5 \cdot g$, which holds due to our assumption.

We note that if we modify the restriction condition from $\frac{g}{2}$ to a smaller linear factor of g, the above proof can be applied similarly; this would allow us to loosen the condition $d \geq 5 \cdot g$ to obtain a slightly smaller constant factor between d and g, if desired. □

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and the introduction summarizes the context of the research and states the main theorems of the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: On the theoretical side, the tightness and not-tightness of the bounds is continuously discussed throughout the theoretical analysis. The conclusion also discusses that the asymptotical optimality of the algorithms might not always imply practical viability at the same time.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The proofs of the results are presented in Section 3 and Appendices A, B, C and D.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [N/A].

Justification: The results in the paper are all theoretical.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [N/A].

   Justification: The results in the paper are all theoretical.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [N/A].

   Justification: The results in the paper are all theoretical.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [N/A].

   Justification: The results in the paper are all theoretical.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [N/A].

   Justification: The results in the paper are all theoretical.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The paper conforms to the Code of Ethics; in particular, as a purely theoretical work on pre-existing models, the paper only has very indirect connections to any of the topics listed in the Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A].

    Justification: This work contributes to the efficient execution of Large Language Models. There are many potential societal consequences of our work and similar works in general, but we cannot foresee any specific immediate concerns related to this specific work.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A].

    Justification: This work only focuses on the more I/O-efficient execution of attention in Large Language Models. As such, the safeguards related to the topic are only the regular safeguards that are required when developing and using Large Language Models.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The only existing dependence is the Approximate Attention algorithm of Alman and Song and its theoretical analysis [4]. This is outlined in the introduction and discussed in more detail in Section 2.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.

- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: No new assets are released in the paper.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: the paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: the paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: No LLMs were involved as important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.