I/O Complexity of Attention, or How Optimal is FlashAttention?

Barna Saha [*1] **Christopher Ye** [*1]

Abstract

Attention is at the heart of the popular Transformer architecture, yet suffers from quadratic time and memory complexity. In a recent significant development, FlashAttention shows that the I/O complexity of attention is the true bottleneck in scaling Transformers. Given two levels of memory hierarchy, a fast cache (e.g. GPU on-chip SRAM) where computation happens and a slow memory (e.g. GPU high-bandwidth memory) where the data resides, the I/O complexity measures the number of accesses to the slow memory. FlashAttention is an I/O-aware algorithm for self-attention that requires $\frac{N^2d^2}{M}$ I/O operations where N is the dimension of the attention matrix, d is the head-dimension and M is the size of cache. Naturally, to further reduce the computational costs of Attention, the authors ask the question: is FlashAttention's I/O complexity optimal for every value of M? We resolve the above question in its full generality by showing an I/O complexity lower bound that matches the upper bound provided by FlashAttention for any values of $M \geq d^2$ within any constant factors. Moreover, our lower bounds do not rely on using combinatorial matrix multiplication for computing the attention matrix: even if one uses fast matrix multiplication, the above I/O complexity bounds cannot be improved. Further, we give a better algorithm with lower I/O complexity for $M < d^2$, and show that it is optimal for combinatorial algorithms. We do so by introducing a new communication complexity protocol for matrix compression, and connecting communication complexity to I/O complexity. We believe this connection could be of independent interest and will find more applications in proving I/O complexity lower bounds in future.

[*]Equal contribution [1]Department of Computer Science and Engineering, University of California San Diego, La Jolla, CA. Correspondence to: Christopher Ye <czye@ucsd.edu>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

1. Introduction

Transformer models (Vaswani et al., 2017) have emerged as the architecture of choice for a variety of applications including natural language processing and computer vision. The self-attention module at the heart of the Transformer architecture requires quadratic time and memory complexity. Thus despite their popularity, Transformers are slow and memory-hungry. In a seminal paper, (Dao et al., 2022) proposes FlashAttention, an IO-aware algorithm for self-attention. Given two levels of memory hierarchy, a fast cache (e.g. GPU on-chip SRAM) where computation happens and a slow memory (e.g. GPU high-bandwidth memory) where the data resides, the I/O complexity measures the number of accesses to the slow memory. (Dao et al., 2022) argues that I/O complexity is indeed the true bottleneck in achieving wall-clock speed up, and provide an algorithm that achieves an I/O complexity of $O(\frac{N^2d^2}{M})$ where N is the dimension of the attention matrix, d is the head-dimension and M is the cache size. In self-attention, three input matrices $Q, K, V \in \mathbb{R}^{N\times d}$ are used to compute $\mathrm{softmax}(QK^T)V$ where $A = \exp(QK^T)$ is the attention matrix. For $M = \Theta(Nd)$, the above I/O complexity bound becomes Nd which matches the trivial lower bound $\Omega(Nd)$, as the algorithm must at least read the input matrices of size $\Theta(Nd)$ into cache and write the output matrix of size $\Theta(Nd)$ to memory. This leaves a tantalizing open question whether FlashAttention I/O complexity is optimal for any values of M (specifically $M = o(Nd)$).

We resolve this question in affirmative for any value of M as long as $M \geq d^2$. Moreover, for $M < d^2$, we give a better algorithm than FlashAttention and also show the bound is optimal among algorithms using combinatorial matrix multiplication. Formally, we give an algorithm (Algorithm 1) computing Attention with I/O complexity $O\left(\frac{N^2d}{\sqrt{M}}\right)$. Note that whenever $M < d^2$, $\frac{N^2d}{\sqrt{M}} \leq \frac{N^2d^2}{M}$ and both algorithms have I/O complexity N^2 when $M = d^2$. Our algorithm matches the optimal I/O complexity of the standard matrix multiplication algorithm on $N \times d$ and $d \times N$ matrices (Hong & Kung, 1981). This is no accident, and in fact we prove that for $M \leq d^2$, the I/O complexity of self-attention is at least as high as the I/O complexity of computing matrix multiplication of two $N \times d$ and $d \times N$ matrices. For $M \geq d^2$ which is also the more practical regime, our proof

of optimality of FlashAttention brings in several new ideas.

First, we show that FlashAttention has optimal I/O complexity among algorithms using combinatorial matrix multiplication by utilizing the famous *red-blue pebble game* from the early eighties (Hong & Kung, 1981). However, it may still be possible to improve FlashAttention by utilizing fast matrix multiplication (Gall & Urrutia, 2018; Williams et al., 2024) to obtain lower I/O complexity. We rule out any such possibilities. We prove a general lower bound against all such algorithms. We introduce a new matrix compression problem, and show that it has a high communication complexity. Next, we establish a connection between the communication complexity of a matrix compression protocol with the I/O complexity of attention. We show even when the input matrix entries are restricted to binary, the lower bound holds within polylogarithmic factors by utilizing Binary BCH codes (Hocquenghem, 1959; Bose & Ray-Chaudhuri, 1960) from coding theory to prove lower bounds against binary matrix compression.

1.1. Related Work

Many papers have studied attention, a key bottleneck of the transformer architecture (Vaswani et al., 2017) which has become the predominant architecture in applications such as natural language processing. Many approximate notions have been studied to reduce its compute and memory constraints (Brown et al., 2020; Katharopoulos et al., 2020; Kitaev et al., 2020; Zaheer et al., 2020; Chen et al., 2021; Choromanski et al., 2021; Alman & Song, 2024; Han et al., 2023). Algorithms such as FlashAttention have made significant practical gains by designing I/O-aware algorithms (Dao et al., 2022; Dao, 2023).

The role of learning under bounded space constraints has been studied primarily in the context of bounded memory, with applications including Online Learning (Srinivas et al., 2022; Peng & Rubinstein, 2023; Peng & Zhang, 2023), continual learning (Chen et al., 2022), convex optimization (Woodworth & Srebro, 2019; Marsden et al., 2022; Chen & Peng, 2023), and others (Raz, 2019; Sharan et al., 2019; Gonen et al., 2020). In this work, we instead consider learning in the context of a *two-level* memory hierarchy with a *bounded cache*. While the algorithm has unlimited memory, it can only access a small portion of this memory at any given time, and is charged for every access to memory. We believe that the task of minimizing I/O complexity is both theoretically interesting and practically significant, as I/O operations often form a key computational bottleneck in practice.

There is a long-line of work on I/O complexity (Hong & Kung, 1981; Aggarwal & Vitter, 1988; Pagh & Silvestri, 2014). Within this body of work, many papers have focused on the problem of matrix multiplication (Ballard et al., 2012b;a; Pagh & Stöckel, 2014; Scott et al., 2015; Bilardi & Stefani, 2017). Communication complexity plays an important role in establishing lower bounds for streaming and cell probe models (Verbin & Zhang, 2013; Hu et al., 2015), and has been used to establish I/O complexity bound for data structure problems (Eenberg et al., 2017).

1.2. Our Contributions

We study the I/O complexity of attention on a two-level memory hierarchy. The attention mechanism takes inputs $Q, K, V \in \mathbb{R}^{N \times d}$ and computes $\mathrm{softmax}(QK^T)V$ (see Section 2.1) where $A = \exp(QK^T)$ is the attention matrix. Our first result resolves the I/O complexity of any algorithm computing attention using the standard matrix multiplication algorithm. This answers an open question of (Dao et al., 2022) and shows that FlashAttention is optimal among this class of algorithms.

Theorem 3.1. *The I/O complexity of attention with standard matrix multiplication is*

$$\Theta\left(\min\left(\frac{N^2 d}{\sqrt{M}}, \frac{N^2 d^2}{M}\right)\right)$$

We divide the analysis into two cases, the large cache ($M \geq d^2$) and small cache ($M < d^2$) case. When $M < d^2$, we show that attention and matrix multiplication are equivalent. In the small cache setting, $\frac{N^2 d^2}{M} > \frac{N^2 d}{\sqrt{M}} > N^2$, so we can explicitly write QK^T to memory while computing attention. Thus, any algorithm computing attention in fact computes QK^T, establishing an equivalence between attention and matrix multiplication.

In the more interesting large cache case, when $M \geq d^2$, the upper bound is given by the breakthrough FlashAttention algorithm (Dao et al., 2022). In this setting, the I/O complexity approaches $O(Nd)$ as cache size increases, providing significant practical improvements, despite the theoretical time complexity remaining $O(N^2 d)$. To prove a matching lower bound, we use the red-blue pebble game of (Hong & Kung, 1981). Executing an algorithm $\mathcal{A}$ on a machine with cache size M is equivalent to playing the red-blue pebble game with M red pebbles on the computational directed acyclic graph G corresponding to $\mathcal{A}$. Specifically, the partitioning lemma (Lemma C.2) states that any successful execution of $\mathcal{A}$ corresponds to a partition of G where each part corresponds to a sub-computation of $\mathcal{A}$ with no I/O operations (called a M-partition). Thus, it suffices to prove a lower bound on the size of any M-partition to obtain a I/O complexity lower bound for $\mathcal{A}$ on a machine with cache size M. Our lower bound on the partition size then follows from a careful analysis of the computational graph of attention (Figure 3). Specifically, we show that any part V_i in the partition can only compute $\frac{M^2}{d^2} + M \leq \frac{2M^2}{d^2}$ entries in the product QK^T.

We then extend our lower bound in the large cache ($M \geq d^2$) case to arbitrary algorithms for attention. More precisely, we show that even with fast matrix multiplication (FMM), $\Omega\left(\frac{N^2d^2}{M}\right)$ I/O operations are required to compute attention. In particular, fast matrix multiplication can only reduce time complexity, not I/O complexity.

Theorem 4.8. *Suppose $Q, K \in \mathbb{F}_q^{N\times d}$ where $q > N$. The I/O complexity of attention (with any matrix multiplication algorithm) is $\Omega\left(\min\left(\frac{N^2d^2}{M}, N^2\right)\right)$.*

To establish a general lower bound, we can no longer reason about a specific computational graph. Instead, we appeal to communication complexity, a useful tool for proving lower bounds for learning (Dagan et al., 2019; Kane et al., 2019), specifically space-bounded learning (Chen et al., 2022).

For simplicity, assume that $\mathcal{A}$ performs I/O operations in batches of size M (Lemma 4.1 shows that this can be assumed without loss of generality). Within each batch, $\mathcal{A}$ can only access the cache of size M, with no further I/O operations. In order to give a lower bound on the number of batches, we argue that $\mathcal{A}$ cannot make too much progress in each batch. In the context of attention, we measure progress as the number of entries computed in QK^T (regardless of whether these entries are written to memory).

Formally, we introduce the B-entry matrix compression problem (Definition 4.2) and argue that any algorithm making too much progress on a given batch gives an efficient communication protocol (Theorem 4.4). We then complete the argument by giving a lower bound on the communication complexity of the matrix compression problem (Lemma 4.6). Our lower bound follows from constructing input matrices satisfying strong linear independence constraints. For large q, this is achieved by Vandermonde matrices.

Finally, we relax the constraint $q > N$ and consider the special case $q = 2$, i.e. Q, K, V are binary matrices and obtain a lower bound tight up to polylogarithmic factors. Our lower bound uses Binary BCH codes (Hocquenghem, 1959; Bose & Ray-Chaudhuri, 1960) to construct matrices satisfying strong linear independence constraints.

Theorem 4.11. *Suppose $Q, K \in \{0,1\}^{N\times d}$. The I/O complexity of attention (with any matrix multiplication algorithm) is $\Omega\left(\min\left(\frac{N^2d^2}{M\log^2 N}, N^2\right)\right)$.*

In the small cache setting ($M < d^2$), Theorem 4.14 gives a simple equivalence between attention and rectangular matrix multiplication.

2. Preliminaries

Given a matrix A, we index an entry as $A[i, j]$. The i-th row is $A[i]$ while the j-th column is $A[*, j]$. A^T, A^{-1} denote the transpose and inverse of A. For $v \in \mathbb{R}^n$, let $\mathrm{diag}(v)$ denote the matrix $D \in \mathbb{R}^{n\times n}$ with $D[i, i] = v[i]$.

2.1. The Attention Mechanism

Given input matrices $Q, K, V \in \mathbb{R}^{N\times d}$, the attention mechanism computes $D^{-1}AV$ where the attention matrix $A = \exp(QK^T)$ is computed by taking exponents entry-wise and $D = \mathrm{diag}(A \cdot \mathbf{1})$ is the diagonal matrix containing row-sums of A.

2.2. The Memory Hierarchy

In this work, we assume that the memory hierarchy has two levels: the small fast layer (called the *cache*) and the large slow layer (called the *memory*). We assume that the memory is unbounded while the cache is bounded by some size constraint M. Furthermore, computations only occur on the cache.

2.3. Red-Blue Pebble Game

We require the red-blue pebble game of (Hong & Kung, 1981), designed to model computations on a two-level memory hierarchy. In this section, we give only the necessary definitions. For a longer discussion, see Appendix C.

Definition 2.1. [Red-Blue Pebble Game (Hong & Kung, 1981)] Let G be a directed acyclic graph with a set of *input* vertices containing all vertices with no parents, and a set of *output* vertices containing all vertices with no children. A *configuration* is a pair of (not necessarily disjoint) subsets of vertices, one containing all vertices with red pebbles, and the other containing all vertices with blue pebbles.

The *initial* (resp. *terminal*) configuration is one in which only input (resp. output) vertices contain pebbles, and all of these pebbles are blue. The rules of the red-blue pebble game are as follows:

- **R1** (Input) A red pebble may be placed on any vertex with a blue pebble.
- **R2** (Output) A blue pebble may be placed on any vertex with a red pebble.
- **R3** (Compute) A red pebble may be placed on a vertex if all its parents have red pebbles.
- **R4** (Delete) A pebble may be removed from any vertex.

A *transition* is an ordered pair of configurations where the second can be obtained from the first following one of the above rules. A *calculation* is a sequence of configurations, where each successive pair forms a transition. A *complete calculation* is a calculation that begins with the initial configuration and ends with the terminal configuration.

To model a bounded cache, we assume that there are at most M red pebbles on the graph at any given time, while any number of blue pebbles can be placed on the graph. Given a graph G, we are interested in the I/O complexity.

Definition 2.2. [I/O Complexity (Hong & Kung, 1981)] Given a graph G and integer M, the I/O complexity $Q(G, M)$ is defined by the minimum number of transitions according to rules **R1** and **R2** required by any complete calculation. When the underlying graph G is clear, we omit G and write $Q(M)$.

To obtain I/O complexity lower bounds, (Hong & Kung, 1981) show that any complete calculation induces a M-partition of G. Before defining the M-partition, we give the necessary definitions of dominator sets and minimum sets.

Definition 2.3 (Dominator Set)**.** Let $S \subset V$. $D \subset V$ is a *dominator set* of S if every path from an input vertex of G to a vertex in S contains a vertex in D.

Definition 2.4 (Minimum Set)**.** Let $S \subset V$. The *minimum set* of S is the subset of vertices in S with no children in S.

We now give the definition of a M-partition (S-partition in (Hong & Kung, 1981)).

Definition 2.5. [M-partition (Hong & Kung, 1981)] Let G be a directed acyclic graph. A family of subsets $\{V_i\}_{i=1}^h$ is a *M-partition* of G if the following properties hold:

- **P1** (Partition) The sets V_i are disjoint and $V = \bigcup_{i=1}^h V_i$.
- **P2** (Dominator) For each V_i, there exists a dominator set D_i of size at most M.
- **P3** (Minimum) For each V_i, the set of minimum vertices M_i has size at most M.
- **P4** (Acyclic) There is no cyclic dependence among vertex sets $\{V_i\}_{i=1}^h$ (Definition C.1).

The partitioning lemma of (Hong & Kung, 1981) shows that it suffices to prove a lower bound on the size of any $2M$-partition to obtain a lower bound on the I/O complexity.

Lemma 2.6 (Lemma 3.1 of (Hong & Kung, 1981))**.** *For any directed acyclic graph G, let $P(G, M)$ denote the minimum size of any M-partition of G. Then,*

$$Q(G, M) \geq M \cdot (P(G, 2M) - 1)$$

2.4. Computational Graph for the Attention Mechanism

Figure 3 gives a computational graph modelling the attention mechanism (Dao et al., 2022; Dao, 2023). In this computational graph, we assume that matrix products are computed using the standard algorithm, that is, we compute $(AB)_{ij} = \sum_k A_{ik}B_{kj}$ for all i, j.

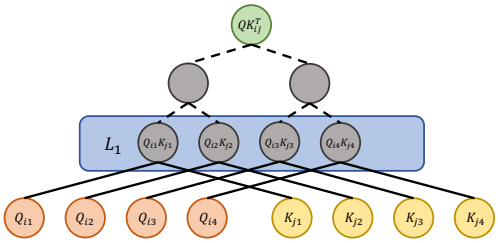

Figure 1. A single summation tree with $d = 4$. Orange and yellow vertices denote inputs from Q, K respectively. Grey and green vertices denote *level-1 vertices*. The green vertex denotes an entry in QK^T. Edges in the bottom layer denote multiplications and edges in the tree denote additions. Vertices in the blue box denote vertices in L_1.

The key structure we exploit are *level-1* vertices, making up the summation trees described in Figure 1. Within each summation tree, the leaves are denoted L_1 vertices, each representing a product $Q_{ik}K_{jk}$. For every entry $(QK^T)_{ij}$, there is a summation tree computing its value, and all N^2 trees are vertex-disjoint. See Appendix A.1 for a detailed discussion of the full computational graph.

3. I/O Complexity of Attention

In this section, we present a tight characterization of the I/O complexity of any algorithm computing attention exactly using the standard matrix multiplication algorithm.

Theorem 3.1. *The I/O complexity of attention with standard matrix multiplication is*

$$\Theta\left(\min\left(\frac{N^2 d}{\sqrt{M}}, \frac{N^2 d^2}{M}\right)\right)$$

At the crossover point $M = d^2$, observe $\frac{N^2 d}{\sqrt{M}} = \frac{N^2 d^2}{M} = N^2$. We define $M \geq d^2$ as the *large cache* regime, and $M < d^2$ as the *small cache* regime. We are primarily interested in the large cache regime, since this is where I/O complexity is sub-quadratic. This is the regime where FlashAttention outperforms standard implementations of attention, since we can avoid writing the entire $N \times N$ matrix QK^T to memory.

3.1. Large Cache: $M = \Omega(d^2)$

For large $M = \Omega(d^2)$, we show that the following result of (Dao et al., 2022) is optimal in terms of I/O complexity.

Theorem 3.2 (Theorem 2 of (Dao et al., 2022))**.** *FlashAttention has I/O complexity $O\left(\frac{N^2 d^2}{M}\right)$.*

To prove a matching lower bound, we bound the number of level-1 vertices in each part of a M-partition. This gives a

lower bound on the size of any partition, implying an I/O complexity lower bound via Lemma 2.6.

Lemma 3.3. *Suppose $M = \Omega(d^2)$ and let $\mathcal{P}$ be a M-partition of the computational graph in Figure 3. Let $V' \in \mathcal{P}$. Then, V' contains at most $O\left(\frac{M^2}{d}\right)$ level-1 vertices.*

Proof. First, since V' has a dominator set D' of at most M, and each summation tree in the computational graph is disjoint, there are at most M level-1 summation trees containing a dominator vertex. Next, since V' has at most M minimum vertices, disjointness again implies that at most M level-1 summation trees contain a minimum vertex. See Figure 2 for an example of V' containing dominator and minimum vertices.

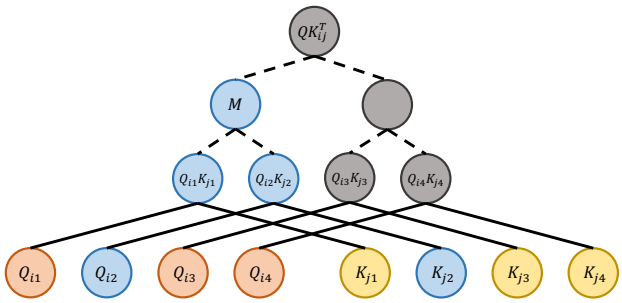

Figure 2. An example of a summation tree containing dominator and minimum vertices. Blue vertices denote elements of the partition V'. $\{M\}$ is the minimum vertex, and $\{Q_{i1}K_{j1}, Q_{i2}, K_{j2}\}$ are the dominator vertices.

Suppose V' contains level-1 vertices not in any of the above $2M$ summation trees (at most M trees containing a dominator and another at most M containing minimum vertices).

Consider a tree T containing level-1 vertices that does not intersect D' or contain a minimum vertex. We denote such a tree as *extra*. Since T does not contain any minimum vertices, the root r of T, representing some entry $(QK^T)[i,j]$, must be in V'. There are $2d$ input vertices with paths to r, namely the inputs $\{Q[i,\ell], K^T[\ell,j]\}_{\ell=1}^d$. We denote the i-inputs of T as the set $\{Q[i,\ell]\}_{\ell=1}^d$ and the j-inputs as the set $\{K^T[\ell,j]\}_{\ell=1}^d$. On each path, all vertices but the inputs are in T. Since T contains no vertices in D', D' must contain all $2d$ input vertices. For example, in Figure 1, V' contains $(QK^T)[i,j]$ and none of the grey vertices, and therefore must contain all $2d = 8$ input vertices (orange and yellow).

For two extra trees $T_{ij}, T_{i'j'}$ whose roots correspond to entries $(QK^T)[i,j], (QK^T)[i',j']$ respectively, observe that the i-inputs are equal if $i = i'$ and disjoint otherwise. Similarly, the j-inputs are equal if $j = j'$ and disjoint otherwise. Suppose V' contains level-1 vertices in C extra level-1 summation trees. The C roots of these trees correspond to C entries in the matrix QK^T. Suppose the C entries have I distinct values in the row index. Choose one entry in C for each row index. Each entry requires d i-input vertices to be placed in the dominator set, and furthermore these input vertices are disjoint. Therefore, there are at least Id input vertices from Q in the dominator set D'. By a similar argument, if the C entries have J distinct values in the column index, there are at least Jd input vertices from K in the dominator set D'. In particular, if there are C extra trees, then there are at least $(I+J)d$ input vertices in the dominator set D'. Finally, we can bound C by observing that $I + J \geq \max(I,J) \geq \sqrt{C}$ since $IJ \geq C$. In particular,

$$\sqrt{C}d \leq (I+J)d \leq |D'| \leq M$$

so that $C \leq \frac{M^2}{d^2}$.

In total, V' contains vertices in at most $C + 2M$ level-1 trees. Each level-1 tree contains $O(d)$ level-1 vertices so that V' contains at most,

$$O\left(\frac{M^2}{d} + Md\right) = O\left(\frac{M^2}{d}\right)$$

level-1 vertices, where we have used $M = \Omega(d^2)$. □

Using Lemma 2.6, we obtain a lower bound for computing attention using standard matrix multiplication. In fact, since we have not used any other part of the computational graph, our lower bound holds for any algorithm computing QK^T using standard matrix multiplication.

Lemma 3.4. *Suppose $M = \Omega(d^2)$. Then, $P(M) = \Omega\left(\frac{N^2d^2}{M^2}\right)$ and $Q(M) = \Omega\left(\frac{N^2d^2}{M}\right)$.*

The (simple) proof is deferred to Appendix A.2.

3.2. Small Cache: $M = o(d^2)$

When $M = o(d^2)$, we show that attention is equivalent to matrix multiplication, establishing a $\Theta\left(\frac{N^2d}{\sqrt{M}}\right)$ bound on the I/O complexity. The details are left to Appendix A.3.

4. I/O Complexity of Attention with Fast Matrix Multiplication

We can in fact lower bound the I/O complexity of any algorithm computing attention exactly, including those using fast matrix multiplication. The only assumption we require is that the algorithm computes the matrix product QK^T explicitly (whether or not it writes the result to memory). Since we do not make any further assumptions on the algorithm, the previous approach of analyzing a computational directed acyclic graph is not sufficient (Hong & Kung, 1981). Instead, we relate I/O complexity to compression lower bounds. As discussed previously, we are primarily interested in the large cache regime where $M = \Omega(d^2)$.

4.1. Large Cache: $M = \Omega(d^2)$

Given some algorithm $\mathcal{A}$ and a sample execution, we split this computation into batches of roughly M I/O operations each using the framework of (Pagh & Silvestri, 2014).

Lemma 4.1. *[Theorem 3 of (Pagh & Silvestri, 2014)] Suppose $\mathcal{A}'$ is an execution of algorithm $\mathcal{A}$ on a machine with cache of size M. The execution $\mathcal{A}'$ can be split into T epochs of at most M I/O operations, such that in each epoch the algorithm $\mathcal{A}$ has access to a cache of size at most $2M$ and no I/O operations. Furthermore, the I/O complexity of $\mathcal{A}'$ is at least $(T-1)M$.*

We provide the proof in Appendix B for completeness. Intuitively, we split the execution $\mathcal{A}'$ into T epochs such that *exactly* M I/O operations take place in each of the first $T-1$ epochs, and at most M I/O operations take place in the final epoch. In other to simulate an epoch, the algorithm uses the extra M entries in the cache of size $2M$ to create a buffer for the next M I/O operations, reading at most M entries into cache at the start of the epoch, and writing at most M entries into memory at the end of the epoch.

To show our lower bound, we will simplify the problem and assume the matrices Q, K, V have entries in some finite field $\mathbb{F}_q$. This is similar to the "indivisibility" assumptions of (Arge & Miltersen, 1998; Pagh & Silvestri, 2014), as the field size fixes some bound on the amount of information in a single cache entry. Otherwise, if each cache entry can hold an arbitrary real number, then even when $M = 1$ we could encode the entire matrices Q, K in a single cache entry.

Under this assumption, we will assume the cache holds M elements of $\mathbb{F}_q$, and we will correspondingly define I/O complexity as the number of finite field elements read and written between the memory hierarchy. In practice, matrices Q, K have arbitrary real entries. Since our lower bounds only need to consider algorithms computing QK^T, we may restrict the inputs to some finite field $\mathbb{F}_q$ without loss of generality (choosing q large enough to avoid overflows under arithmetic operations) and we can assume q is of polynomial size since we do not need to consider the softmax operation. We will separately consider the cases $q = 2$ (i.e. every entry in the cache and the matrices is a single bit) and the case for an arbitrarily large finite field of size q.

4.2. I/O Complexity and Compression

We will prove our I/O lower bound by arguing that any algorithm computing entries of QK^T amounts to an efficient compression protocol. First, we show a lower bound for any compression protocol computing entries of QK^T.

Definition 4.2 (Matrix Compression)**.** Let $B \geq 0$. In the B-entry matrix compression problem $\mathbf{MatrixEntryCompression}_B$ Alice is given input matrices $Q, K \in \mathbb{F}_q^{N \times d}$. Alice must send a message to Bob so that Bob can compute at least B entries in QK^T.

We review the definition of one-way communication complexity below.

Definition 4.3 (One-Way Communication Complexity)**.** Let $f : \mathcal{X}_A \times \mathcal{X}_B \to \mathcal{Y}$ be an arbitrary function. Suppose Alice has $x \in \mathcal{X}_A$ and Bob has $y \in \mathcal{X}_B$. A *one-way communication protocol* computing f is a pair of functions (E, D) such that $D(E(x), y) = f(x, y)$ for all $(x, y) \in \mathcal{X}_A \times \mathcal{X}_B$. The *complexity* of the protocol is $\max_{(x,y)} |E(x)|$. The *one-way communication complexity* of f is the complexity of the optimal protocol.

$$\mathbf{OneWayCC}(f) = \min_{(E,D)} \max_{(x,y) \in \mathcal{X}_A \times \mathcal{X}_B} |E(x)|$$

The one-way communication complexity of $\mathbf{MatrixEntryCompression}_B$ is $O(B \log q)$, since Alice can compute QK^T and transmit the relevant B entries. Instead, if Bob computes a square sub-matrix of QK^T, Alice can instead send the relevant bits of Q, K, transmitting only $O(d\sqrt{B} \log q)$ entries. We give a lower bound of $\Omega\left(\min(d\sqrt{B} \log q, B \log q)\right)$, showing that this is essentially tight.

Our key lemma states that any algorithm outputting many bits of QK^T in one epoch (as described in Lemma 4.1) gives an efficient compression protocol for $\mathbf{MatrixEntryCompression}_B$.

Theorem 4.4. *Let $\mathcal{A}'(Q, K)$ be an execution of an algorithm $\mathcal{A}$ on input Q, K on a machine with cache of size M. Let B_t be the number of entries of QK^T computed in the t-th epoch as described in Lemma 4.1 and $B'(Q, K) = \max_t B_t$. Define,*

$$B^* = \min_{Q,K} B'(Q, K)$$

so that on every input $\mathcal{A}$ computes at least B^* entries of QK^T on some epoch. Then, the one-way communication complexity of $\mathbf{MatrixEntryCompression}_{B^*}$ is at most $2M \log q$.

Proof. We will use $\mathcal{A}$ to construct a compression protocol. Given input matrices Q, K, we execute $\mathcal{A}$ on Q, K to obtain an execution $\mathcal{A}'$. As described in Lemma 4.1, the execution $\mathcal{A}'$ can be split into epochs, where in each epoch the algorithm $\mathcal{A}$ has access only to $2M$ finite field elements in cache and reads no other inputs from memory in this epoch. Let t^* be an epoch in which the algorithm $\mathcal{A}$ computes $B'(Q, K)$ entries of the product QK^T. In particular, Alice can send to Bob the state of the cache of size at most $2M$ at the beginning of the t^*-th epoch, so that Bob computes $B'(Q, K) \geq B^*$ entries of QK^T. □

4.3. Matrix Compression Lower Bounds

In this section, we prove a lower bound on the one-way communication complexity of the B-entry $\mathbf{MatrixEntryCompression}$ problem. More precisely, we will show an upper bound on the number of entries that can be computed given a message of size M. Our previous discussion then implies an upper bound on the number of bits computed in each epoch, therefore lower bounding the number of epochs and the I/O complexity.

As a warmup, we give a simple lower bound of $\sqrt{B}\log q$.

Lemma 4.5. *Let $B \geq 0$. Then, the one-way communication complexity of $\mathbf{MatrixEntryCompression}_B$ is at least $\sqrt{B}\log q$.*

The proof is presented in Appendix B. We generalize this for matrices Q, K of dimension $N \times d$ with rank $d \geq 1$.

Lemma 4.6. *Suppose $Q, K \in \mathbb{F}_q^{N\times d}$ with finite field $\mathbb{F}_q$ of size $q > N$. Then, the one-way communication complexity of $\mathbf{MatrixEntryCompression}_B$ is at least $\min\left(d\sqrt{\frac{B}{2}}, \frac{B}{4}\right)\log q$.*

We show that for any set of B indices, there are more than q^M possible values. Thus, a message length of at least $M\log q$ is required to specify these entries exactly.

Proof. Let M denote the maximum message length in the communication protocol. Let $I \subseteq [N]^2$ denote the indices of QK^T computed with $B = |I|$. Define R_I to be the distinct row indices in I and C_I to be the distinct column indices in I so that $I \subseteq R_I \times C_I$.

For each $i \in R_I$, let $R_i = \{j \text{ s.t. } (i,j) \in I\}$ be the computed entries in the i-th row of QK^T. Similarly, for each $j \in C_I$, let $C_j = \{i \text{ s.t. } (i,j) \in I\}$ be the computed entries in the j-th column of QK^T. Next, define $L_R = \{i \text{ s.t. } |R_i| \geq d\}$ and $L_C = \{j \text{ s.t. } |C_j| \geq d\}$. We also define $S_R = \{(i,j) \in I \text{ s.t. } i \notin L_R\}$ and $S_C = \{(i,j) \in I \text{ s.t. } j \notin L_C\}$.

First, suppose $\max(|L_R|, |L_C|) \geq \sqrt{B/2}$. Without loss of generality, assume $|L_R| \geq \sqrt{B/2}$. Since $q > N$, we fix K to be the Vandermonde matrix guaranteed by Lemma 4.7. In particular, every subset of d columns in K^T is linearly independent.

Fix some row $i \in R_I$. We claim that there are at least $q^{\min(|R_i|,d)}$ distinct values in the R_i indices of the i-th row. First, suppose $|R_i| \leq d$. By the construction of matrix K and $|R_i| \leq d$, the columns of K^T indexed by R_i are linearly independent. Then, for any $\vec{v} \in \mathbb{F}_q^{|R_i|}$, we can set the i-th row of Q to be,

$$Q[i] = \vec{v}\left(K[r_1]^T K[r_2]^T \dots K[r_{|R_i|}]^T\right)^{-1}$$

where $K[r_i]$ is the r_i-th row of K and $R_i = \{r_1, \dots, r_{|R_i|}\}$. In particular, this choice of $Q[i]$ ensures that the R_i entries of QK^T are exactly $\vec{v}$ so that there are at least $q^{|R_i|}$ distinct values. Whenever $|R_i| > d$, we simply take an arbitrary subset of R_i of size d, and use their linear independence to proceed with the same argument, thus obtaining the lower bound of $q^{\min(|R_i|,d)}$.

Finally, note that the we can obtain these distinct values for each row in R_I independently, since we have fixed K as the Vandermonde matrix and for each row we only modify the entries of $Q[i]$. In particular, the total number of possible distinct values in all the entries of I is at least,

$$\prod_{i\in R_I} q^{\min(|R_i|,d)}$$

so that we obtain the following lower bound on the message length of the communication protocol in order to unambiguously specify B entries,

$$\begin{aligned} M &\geq \left(\sum_{i\in R_I} \min(|R_i|, d)\right)\log q \\ &\geq (d|L_R| + |S_R|)\log q \\ &\geq d\sqrt{\frac{B}{2}}\log q \end{aligned} \tag{1}$$

where $S_R = \{(i,j) \in I \text{ s.t. } i \notin L_R\}$ and the final inequality follows from our assumption on L_R. In particular, this implies $M \geq d\sqrt{B/2}\log q$ as desired.

Finally, we consider the case $\max(|L_R|, |L_C|) < \sqrt{B/2}$. Then, the number of entries in $L_R \times L_C$ is at most $\frac{B}{2}$. Note that any entry of I not in $L_R \times L_C$ must be in $S_R \cup S_C$ and therefore,

$$\frac{B}{2} \leq |S_R| + |S_C|$$

Assume without loss of generality $|S_R| \geq |S_C|$. Equation (1) then implies $M \geq \frac{B}{4}\log q$. □

In our lower bound construction, we require matrices satisfying strong linear independence constraints. Specifically, we require a $N \times d$ matrix such that every subset of d rows is linearly independent. When the matrices have elements in a large finite field, this is obtained by Vandermonde matrices.

Lemma 4.7. *There is a $N \times d$ Vandermonde matrix with entries in a finite field $\mathbb{F}_q$ of size $q > N$ where every subset of d rows is linearly independent.*

We leave the proof to Appendix B. We are now ready to prove the main result of this section. For $M \geq d^2$, this matches the upper bound given by (Dao et al., 2022).

Theorem 4.8. *Suppose $Q, K \in \mathbb{F}_q^{N\times d}$ where $q > N$. The I/O complexity of attention (with any matrix multiplication algorithm) is $\Omega\left(\min\left(\frac{N^2d^2}{M}, N^2\right)\right)$.*

Proof. The theorem follows from combining Theorem 4.4 and Lemmas 4.1 and 4.6. Consider an arbitrary algorithm $\mathcal{A}$ and its best execution $\mathcal{A}'$, on an input Q, K, V with $B'(Q, K) = B^*$ as described in Theorem 4.4.

Since $q > N$ we apply Lemmas 4.1 and 4.6 and obtain

$$\min\left(d\sqrt{\frac{B^*}{2}}, \frac{B^*}{4}\right)\log q \leq 2M \log q$$

In particular, the maximum number of entries of QK^T computed in any epoch has the upper bound,

$$B^* = O\left(\max\left(\frac{M^2}{d^2}, M\right)\right)$$

Then since the algorithm computes all N^2 values of QK^T (regardless of whether these values are written to memory from cache), the number of epochs is at least,

$$T = \Omega\left(\min\left(\frac{N^2d^2}{M^2}, \frac{N^2}{M}\right)\right)$$

Then, since the I/O complexity of $\mathcal{A}$ is at least $(T-1)M$, this completes the lower bound. □

Whenever $M \geq d^2$, we obtain a lower bound matching the $O\left(\frac{N^2d^2}{M}\right)$ algorithm of (Dao et al., 2022).

4.4. Binary Matrix Compression Lower Bounds

In the previous section, we obtained a tight lower bound for the I/O complexity of attention when the entries are allowed to come from a large finite field. We believe it is an interesting theoretical question to investigate the I/O complexity with entries in smaller finite fields. Specifically, we will consider the binary finite field $\mathbb{F}_2 = \{0, 1\}$.

The assumption $q > N$ was only required to construct a matrix satisfying strong linear independence constraints in Lemma 4.7. Even relaxing this constraint and considering $q = 2$, we can still construct matrices satisfying fairly strong linear independence constraints using error correcting codes. In fact, our previous use of Vandermonde matrices can be interpreted as using Reed-Solomon codes by allowing for arbitrarily large finite fields. Recall that a linear code $\mathcal{C} \subset \{0, 1\}^N$ is the null-space of the parity check matrix H. Using Binary BCH codes, we can obtain a similar result with binary matrices.

Lemma 4.9. *There exists a matrix $K \in \{0, 1\}^{N\times d}$ such that every set of $\frac{2d}{\log(N+1)} - 1$ rows is linearly independent.*

Given the construction of the matrix K, we can prove the following analogues of Lemma 4.6 and Theorem 4.8. These results match the upper bound of Theorem 3.2 up to a $O(\log^2 N)$ factor.

Lemma 4.10. *Suppose Q, K are binary $N \times d$ matrices. Then, the one-way communication complexity of $\mathbf{MatrixEntryCompression}_B$ is at least $\Omega\left(\min\left(\frac{d\sqrt{B}}{\log N}, B\right)\right)$.*

We now state the I/O complexity lower bound of attention given binary input matrices.

Theorem 4.11. *Suppose $Q, K \in \{0, 1\}^{N\times d}$. The I/O complexity of attention (with any matrix multiplication algorithm) is $\Omega\left(\min\left(\frac{N^2d^2}{M\log^2 N}, N^2\right)\right)$.*

We leave the proofs of this section to Appendix B.1.

4.5. Small Cache: $M = o(d^2)$

In the small cache setting, we proved an equivalence between attention and matrix multiplication in the setting where both are computed using the standard algorithm. We do the same for algorithms using fast matrix multiplication.

Let $Q_{Att}(M)$ denote the I/O complexity of attention on a machine with cache size M. Let $Q_{\mathcal{M}(a,b,c)}(M)$ denote the I/O complexity of multiplying a $a \times b$ matrix with a $b \times c$ matrix on a machine with cache size M. First, we show matrix multiplication is more expensive than attention.

Lemma 4.12. *For all M,*

$$Q_{Att}(M) = O\left(Q_{\mathcal{M}(N,d,N)}(M) + Q_{\mathcal{M}(N,N,d)}(M)\right)$$

When $M \leq d^2$, the two are equivalent.

Lemma 4.13. *Let $M \leq d^2$. Then, $Q_{Att}(M) = \Omega\left(Q_{\mathcal{M}(N,d,N)}(M)\right)$.*

We defer the proofs to Appendix B.2. As a corollary, we obtain the following equivalence in the small cache setting.

Theorem 4.14. *Let $M \leq d^2$. Then,*

$$Q_{Att}(M) = \Omega\left(Q_{\mathcal{M}(N,d,N)}(M)\right)$$
$$Q_{Att}(M) = O\left(Q_{\mathcal{M}(N,d,N)}(M) + Q_{\mathcal{M}(N,N,d)}(M)\right)$$

5. Conclusion

We have established tight I/O complexity lower bounds for attention. Our lower bound in fact holds for any algorithm computing matrix product QK^T. We give a tight characterization of the I/O complexity of algorithms using standard matrix multiplication, answering an open question of (Dao et al., 2022).

Furthermore, in the regime of practical interest, where cache size $M \geq d^2$ is large, we extend our lower bound to algorithms using fast matrix multiplication, showing that FlashAttention is optimal even when fast matrix multiplication is allowed. Furthermore, our results continue a line

of work establishing a connection between communication complexity and I/O complexity (Iacono & Puatracscu, 2011; Eenberg et al., 2017), exhibiting the versatility of this idea. While our lower bounds are tight when the input matrices have entries chosen from a sufficiently large field, our lower bounds lose polylogarithmic factors if the inputs are constrained to be binary matrices. We leave the exact characterization of attention with binary input matrices as an interesting open problem.

Finally, we leave the problem of establishing tight I/O complexity bounds for attention in the small cache regime ($M \leq d^2$) (equivalently rectangular matrix multiplication) as an interesting open problem.

Acknowledgements

The authors are partially supported by NSF grants 1652303, 1909046, 2112533, and EnCORE:HDR TRIPODS Phase II grant 2217058. The authors would like to thank Arya Mazumdar and Harry Sha for helpful discussions, and Kasper Green Larsen for helpful comments. The authors would also like to thank anonymous reviewers for their careful reviews and thoughtful comments.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work. First, we prove FlashAttention is optimal for certain ranges of memory. FlashAttention has been shown to have a huge impact in the scalability of Transformers which is the architecture of choice for a variety of practical applications. Second, we give a better algorithm than FlashAttention when fast memory is small. This can have huge impact especially when resources are low. Third, the techniques that we introduced can be utilized beyond attention and Transformers and may leave a lasting impact in the field of Machine Learning.

References

Aggarwal, A. and Vitter, J. S. The input/output complexity of sorting and related problems. *Commun. ACM*, 1988.

Alman, J. and Song, Z. Fast attention requires bounded entries. *Advances in Neural Information Processing Systems*, 36, 2024.

Arge, L. and Miltersen, P. B. On showing lower bounds for external-memory computational geometry problems. In *External Memory Algorithms, Proceedings of a DIMACS Workshop*, 1998.

Ballard, G., Demmel, J., Holtz, O., Lipshitz, B., and Schwartz, O. Graph expansion analysis for communication costs of fast rectangular matrix multiplication. In *Mediterranean Conference on Algorithms MedAlg*, 2012a.

Ballard, G., Demmel, J., Holtz, O., and Schwartz, O. Graph expansion and communication costs of fast matrix multiplication. *J. ACM*, 59(6):32:1–32:23, 2012b.

Bilardi, G. and Stefani, L. D. The I/O complexity of strassen's matrix multiplication with recomputation. In *Algorithms and Data Structures WADS*, 2017.

Bose, R. C. and Ray-Chaudhuri, D. K. On A class of error correcting binary group codes. *Inf. Control.*, 3(1):68–79, 1960.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In *Neural Information Processing Systems, NeurIPS*, 2020.

Chen, B., Dao, T., Winsor, E., Song, Z., Rudra, A., and Ré, C. Scatterbrain: Unifying sparse and low-rank attention. In *Neural Information Processing Systems, NeurIPS*, 2021.

Chen, X. and Peng, B. Memory-query tradeoffs for randomized convex optimization. In *Symposium on Foundations of Computer Science, FOCS*, 2023.

Chen, X., Papadimitriou, C. H., and Peng, B. Memory bounds for continual learning. In *Symposium on Foundations of Computer Science, FOCS*, 2022.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlós, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. Rethinking attention with performers. In *International Conference on Learning Representations, ICLR*, 2021.

Dagan, Y., Kur, G., and Shamir, O. Space lower bounds for linear prediction in the streaming model. In *Conference on Learning Theory, COLT*, 2019.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Neural Information Processing SystemsNeurIPS*, 2022.

Eenberg, K., Larsen, K. G., and Yu, H. Decreasekeys are expensive for external memory priority queues. In *Symposium on Theory of Computing, STOC*, 2017.

Gall, F. L. and Urrutia, F. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In Czumaj, A. (ed.), *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pp. 1029–1046. SIAM, 2018.

Gonen, A., Lovett, S., and Moshkovitz, M. Towards a combinatorial characterization of bounded-memory learning. In *Neural Information Processing Systems, NeurIPS*, 2020.

Han, I., Jayaram, R., Karbasi, A., Mirrokni, V., Woodruff, D. P., and Zandieh, A. Hyperattention: Long-context attention in near-linear time. *CoRR*, abs/2310.05869, 2023.

Hocquenghem, A. Codes correcteurs d'erreurs. *Chiffers*, 2: 147–156, 1959.

Hong, J.-W. and Kung, H.-T. I/o complexity: The red-blue pebble game. In *Symposium on Theory of Computing STOC*, 1981.

Hu, X., Tao, Y., Yang, Y., Zhang, S., and Zhou, S. On the I/O complexity of dynamic distinct counting. In *International Conference on Database Theory, ICDT*, 2015.

Iacono, J. and Puatracscu, M. Using hashing to solve the dictionary problem (in external memory). *CoRR*, abs/1104.2799, 2011.

Kane, D., Livni, R., Moran, S., and Yehudayoff, A. On communication complexity of classification problems. In *Conference on Learning Theory, COLT*, 2019.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning, ICML*, 2020.

Kitaev, N., Kaiser, L., and Levskaya, A. Reformer: The efficient transformer. In *International Conference on Learning Representations, ICLR*, 2020.

Marsden, A., Sharan, V., Sidford, A., and Valiant, G. Efficient convex optimization requires superlinear memory. In *Conference on Learning Theory, COLT*, 2022.

Pagh, R. and Silvestri, F. The input/output complexity of triangle enumeration. In *Principles of Database Systems PODS*, 2014.

Pagh, R. and Stöckel, M. The input/output complexity of sparse matrix multiplication. In *European Symposium on Algorithms ESA*, 2014.

Peng, B. and Rubinstein, A. Near optimal memory-regret tradeoff for online learning. In *Symposium on Foundations of Computer Science, FOCS*, 2023.

Peng, B. and Zhang, F. Online prediction in sub-linear space. In *Symposium on Discrete Algorithms, SODA*, 2023.

Raz, R. Fast learning requires good memory: A time-space lower bound for parity learning. *J. ACM*, 66(1):3:1–3:18, 2019.

Scott, J., Holtz, O., and Schwartz, O. Matrix multiplication i/o-complexity by path routing. In *Symposium on Parallelism in Algorithms and Architectures SPAA*, 2015.

Sharan, V., Sidford, A., and Valiant, G. Memory-sample tradeoffs for linear regression with small error. In *Symposium on Theory of Computing, STOC*, 2019.

Srinivas, V., Woodruff, D. P., Xu, Z., and Zhou, S. Memory bounds for the experts problem. In *Symposium on Theory of Computing, STOC*, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Neural Information Processing Systems NeurIPS*, 2017.

Verbin, E. and Zhang, Q. The limits of buffering: A tight lower bound for dynamic membership in the external memory model. *SIAM J. Comput.*, 42(1):212–229, 2013.

Williams, V. V., Xu, Y., Xu, Z., and Zhou, R. New bounds for matrix multiplication: from alpha to omega. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 3792–3835. SIAM, 2024.

Woodworth, B. E. and Srebro, N. Open problem: The oracle complexity of convex optimization with limited memory. In *Conference on Learning Theory, COLT*, 2019.

Zaheer, M., Guruganesh, G., Dubey, K. A., Ainslie, J., Alberti, C., Ontañón, S., Pham, P., Ravula, A., Wang, Q., Yang, L., and Ahmed, A. Big bird: Transformers for longer sequences. In *Neural Information Processing Systems, NeurIPS*, 2020.

A. Attention with Standard Matrix Multiplication

In this appendix, we provide supplemental material for Section 3.

A.1. Computational Graph for Attention

We provide the computational graph of attention using standard matrix multiplication.

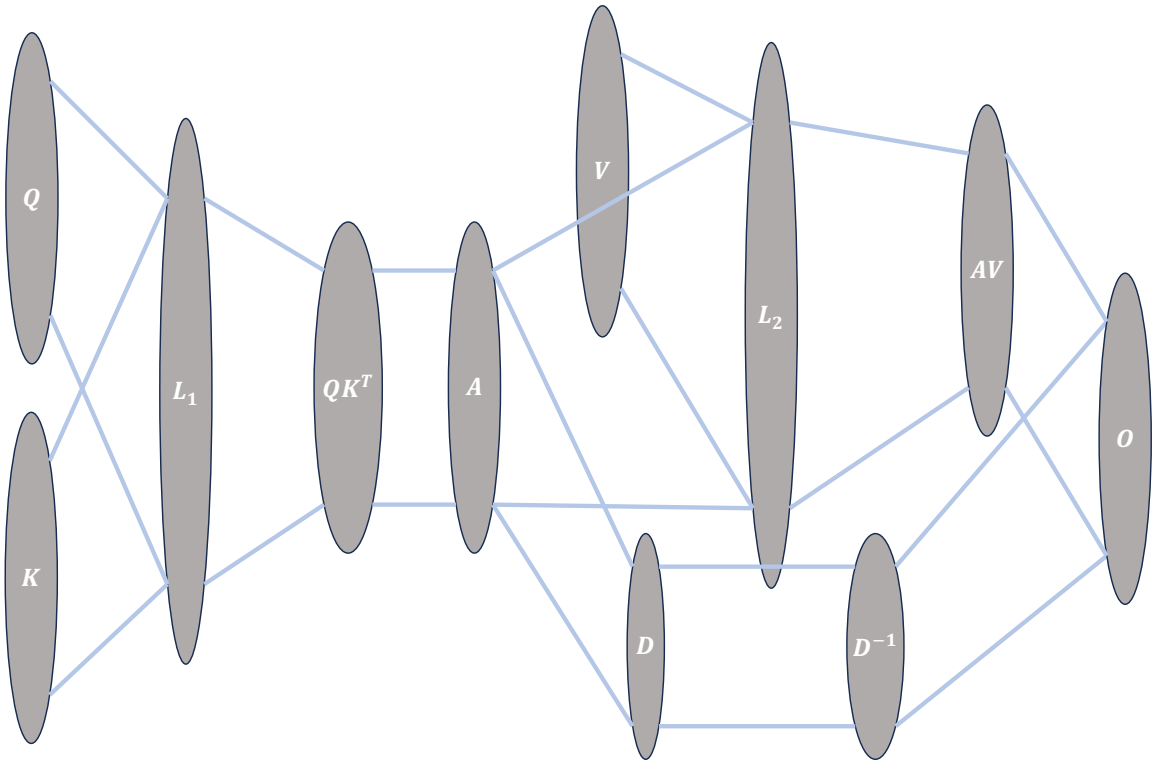

Figure 3. Computational Graph for Attention.

The attention mechanism begins by computing the matrix product QK^T. In the vertex set L_1, there are N^2d vertices representing the values $\{Q_{i\ell}K^T_{\ell j}\}_{i,j,\ell}$. Then, each entry $(QK^T)_{ij}$ is computed by summing the appropriate vertices in L_1, i.e. the node $(QK^T)_{ij} = \sum_\ell Q_{i\ell}K^T_{\ell j}$. In particular, each entry $(QK^T)_{ij}$ is connected to L_1 via a summation tree. Note that all nodes in the summation trees are disjoint. The summation tree can be thought of as a balanced binary tree with d leaves, although the exact structure of the tree (i.e. order of summation) can be arbitrary. For a more detailed description of the computational graph of the standard matrix multiplication algorithm, see (Hong & Kung, 1981). We define the set of *level-1 vertices* to be all nodes in the computational graph that are in vertex sets L_1, QK^T, or one of the intermediate nodes in the N^2 summation trees between the two layers. See Figure 1 illustrates an example summation tree.

Then, each entry in A is computed by taking the exponent of the corresponding entry in QK^T. Each node in D is computed by summing over the rows of A. As above, this is realized in the graph by N disjoint summation trees and D^{-1} is computed by taking the multiplicative inverse of each element in D. The matrix product is computed as in the first step, first by storing all N^2d products in the intermediate layer $L_2 = \{A_{ik}V_{kj}\}_{i,k,j}$ and computing AV via Nd disjoint summation trees. Finally, each node in O is computed by scaling each entry of AV by the appropriate factor in D^{-1}.

Although we have (roughly) described the computational graph of FlashAttention-2 (Dao, 2023) and there are different ways to implement attention, all algorithms begin by computing the product QK^T, and our lower bounds will hold for any algorithm computing this matrix product.

A.2. The Large Cache Case

We give the omitted proof of Lemma 3.4.

Lemma 3.4. *Suppose $M = \Omega(d^2)$. Then, $P(M) = \Omega\left(\frac{N^2d^2}{M^2}\right)$ and $Q(M) = \Omega\left(\frac{N^2d^2}{M}\right)$.*

Proof. Note that there are $\Omega(N^2d)$ level-1 vertices. Since each part in the M-partition contains at most $O\left(\frac{M^2}{d}\right)$ level-1 vertices, the lower bound on the number of parts in the M-partition follows immediately from Lemma 3.3. □

A.3. The Small Cache Case

For $M = o(d^2)$, we first show that there is an algorithm with I/O complexity $O\left(\frac{N^2d}{\sqrt{M}}\right)$.

We define some helpful notation for the algorithm. Given a matrix $A \in \mathbb{R}^{n\times m}$, we index an individual entry as $A[i,j]$. $A[i_1 : i_2, j_1 : j_2]$ denotes a block of A consisting of entries (i,j) where $i \in [i_1, i_2]$ and $j \in [j_1, j_2]$. Given a block size B, the block $A[(i-1)\cdot B + 1 : i \cdot B, (j-1)\cdot B + 1 : j \cdot B]$ is denoted $A^{(B)}[i,j]$. For a vector $v \in \mathbb{R}^n$, we similarly denote entries $v[i]$, a contiguous block of entries as $v[i_1 : i_2]$, and the i-th block of size B as $v^{(B)}[i]$.

Theorem A.1. *There is an algorithm computing attention with I/O complexity $O\left(\frac{N^2d}{\sqrt{M}}\right)$. Furthermore, this algorithm uses $O\left(N^2d\right)$ time and $O\left(Nd + N^2\right)$ space.*

High Level Overview Our algorithm follows standard techniques for reducing the I/O complexity of matrix multiplication. In Phase 1, we compute $A = \exp(QK^T)$ and $D = \mathrm{diag}(A \cdot \mathbf{1})$. First, we divide Q, K into $\frac{N}{\sqrt{M}} \times \frac{d}{\sqrt{M}}$ blocks of size $\sqrt{M} \times \sqrt{M}$ and proceed to compute QK^T one $\sqrt{M} \times \sqrt{M}$ size block at a time. In Lines 5-8, we iterate over blocks of QK^T. In Lines 9-13, we compute each block of QK^T by summing over $\frac{d}{\sqrt{M}}$ block matrix products. After computing QK^T, we apply exponents entry-wise and sum over rows in Lines 14-18 to compute the relevant blocks of A, D.

In Phase 2, we compute the matrix product $D^{-1}AV$, storing the output in matrix O. We imagine D^{-1} as a vector and partition into $\frac{N}{\sqrt{M}}$ blocks of size $\sqrt{M}$ and partition A into $\frac{N}{\sqrt{M}} \times \frac{N}{\sqrt{M}}$ blocks of size $\sqrt{M} \times \sqrt{M}$. Lines 21-24 iterates over blocks of O. Lines 25-29 compute a $\sqrt{M} \times \sqrt{M}$ block of O by summing over $\frac{N}{\sqrt{M}}$ matrix products and scaling by the approprite block of D^{-1}. This completes the overview of the algorithm.

Correctness of Algorithm 1. We begin with Phase 1, showing that each block $A^{(B)}[i,j]$ is computed correctly. Fix a block $A^{(B)}[i,j]$. For any indices $i' \in [(i-1)B+1, iB]$ and $j' \in [(j-1)B+1, jB]$,

$$A[i',j'] = \sum_{\ell'=1}^{d} Q[i,\ell']K^T[\ell',j] = \sum_{\ell=1}^{\lceil d/B\rceil} \sum_{\ell'=(\ell-1)B+1}^{\ell B} Q[i,\ell']K^T[\ell',j]$$

In Line 11, we iterate over i', j', ℓ', adding $Q[i',\ell']K^T[\ell',j']$ to $A[i',j']$. After iterating over $1 \le \ell \le \lceil d/B\rceil$, $A^{(B)}[i,j]$ is computed so that after applying $\exp$ entry-wise, we write the correct block $A^{(B)}[i,j]$ into memory. Next, since d should contain row-sums,

$$d[i'] = \sum_{j'=1}^{N} A[i',j'] = \sum_{j=1}^{\lceil N/B\rceil} \sum_{j'=(j-1)B+1}^{jB} A[i',j']$$

we correctly write $d^{(B)}[i]$ into memory. Throughout Phase 1, the number of items in cache is $3B^2 + B \le 4B^2 \le M$.

Now, we proceed to Phase 2. Let $D = \mathrm{diag}(d) = \mathrm{diag}(A\cdot\mathbf{1})$. Similarly, for $i' \in [(i-1)B+1, iB]$, $j' \in [(j-1)B+1, jB]$

$$O[i',j'] = \frac{1}{D[i',i']}\sum_{k'=1}^{N} A[i',k']V[k',j'] = \sum_{k=1}^{\lceil N/B\rceil} \sum_{k'=(k-1)B+1}^{kB} \frac{A[i',k']V[k',j']}{D[i',i']}$$

which is computed by the loop in Phase 2. In particular, Line 27 adds the appropriate value for each block k. The overall size of cache required is $B + 3B^2 \le 4B^2 \le M$. Thus, Algorithm 1 correctly computes $O = D^{-1}AV$ with cache size M. □

Algorithm 1 SQUARETILINGATTENTION(Q, K, V, M)

Input: Matrices $Q, K, V \in \mathbb{R}^{N \times d}$, Cache size M
Output: $D^{-1}AV$ where $A = \exp(QK^T)$ and $D = \mathrm{diag}(A \cdot \mathbf{1})$
$B \leftarrow \left\lfloor \sqrt{M/4} \right\rfloor$
Phase 1: Compute D, A
for $i = 1$ **to** $\lceil N/B \rceil$ **do**
  Initialize $d^{(B)}[i] \leftarrow 0^B$ in cache
  for $j = 1$ **to** $\lceil N/B \rceil$ **do**
    Initialize $A^B[i,j] \leftarrow 0^{B \times B}$ in cache
    for $\ell = 1$ **to** $\lceil d/B \rceil$ **do**
      Read $Q^{(B)}[i,\ell]$ and $(K^T)^{(B)}[\ell, j]$ into cache
      Compute $A^B[i,j] \leftarrow A^B[i,j] + Q^{(B)}[i,\ell](K^T)^{(B)}[\ell,j]$ in cache
      Delete $Q^{(B)}[i,\ell]$ and $(K^T)^{(B)}[\ell,j]$ from cache
    end for
    Compute $A^B[i,j] \leftarrow \exp(A^B[i,j])$ and write $A^B[i,j]$ into memory
    Compute $d^{(B)}[i] \leftarrow d^{(B)}[i] + A^B[i,j] \cdot \mathbf{1}$ in cache
    Delete $A^B[i,j]$ from cache
  end for
  Write $d^{(B)}[i]$ to memory and delete $d^{(B)}[i]$ from cache
end for
Phase 2: Compute $D^{-1}AV$
for $i = 1$ **to** $\lceil N/B \rceil$ **do**
  Read $d^{(B)}[i]$ into cache
  for $j = 1$ **to** $\lceil d/B \rceil$ **do**
    Initialize $O^{(B)}[i,j] \leftarrow 0^{B \times B}$ in cache
    for $k = 1$ **to** $\lceil N/B \rceil$ **do**
      Read $A^{(B)}[i,k]$ and $V^{(B)}[k,j]$ into cache
      Compute $O^{(B)}[i,j] \leftarrow O^{(B)}[i,j] + \mathrm{diag}\left(d^{(B)}[i]\right)^{-1} A^{(B)}[i,k]V^{(B)}[k,j]$
      Delete $A^{(B)}[i,k]$ and $V^{(B)}[k,j]$ from cache
    end for
    Write $O^{(B)}[i,j]$ to cache and delete $O^{(B)}[i,j]$ from cache
  end for
  Delete $d^{(B)}[i]$ from cache
end for

I/O Complexity of Algorithm 1. In Phase 1, for each iteration through i, j, ℓ, the algorithm reads $O(B^2)$ values from memory into cache. This dominates the I/O complexity of the algorithm. The I/O complexity of Phase 1 is therefore $O\left(\frac{N^2 d}{B^3} B^2\right) = O\left(\frac{N^2 d}{B}\right) = O\left(\frac{N^2 d}{\sqrt{M}}\right)$.

Similarly for Phase 2, the I/O complexity is dominated by the reading A, V into the cache and this has I/O complexity $O\left(\frac{N^2 d}{\sqrt{M}}\right)$, thus bounding the overall I/O complexity. □

Time and Space Complexity of Algorithm 1. Since we use standard matrix multiplication, the overall time complexity is $O(N^2 d)$. The space required is $O(Nd + N^2)$ as the algorithm stores matrices Q, K, V, M and the vector d. □

We now show that this is tight for $M \leq d^2$. We proceed by a reduction to the I/O complexity of matrix multiplication, invoking the following result.

Lemma A.2 (Corollary 6.2 of (Hong & Kung, 1981))**.** *Let $A \in \mathbb{R}^{m \times k}$ and $B \in \mathbb{R}^{k \times n}$. The standard algorithm for matrix multiplication satisfies $Q(M) = \Omega\left(\frac{mkn}{\sqrt{M}}\right)$.*

Theorem A.3. *Suppose $M = o(d^2)$. Then, the I/O complexity of attention using standard matrix multiplication is at least*

$\Omega\left(\frac{N^2 d}{\sqrt{M}}\right)$

Proof. The lower bound follows from a reduction to matrix multiplication. We take advantage of the fact that if $M \leq d^2$, $\frac{N^2 d}{\sqrt{M}} \geq N^2$, so the algorithm can afford to write the attention matrix A explicitly to memory. Given an algorithm $\mathcal{A}$ for attention, we have the following algorithm for matrix multiplication. Given inputs Q, K, we execute $\mathcal{A}$ with one modification: whenever an entry of QK^T is computed for the first time, write this entry to memory. Over the course of the algorithm, this computes QK^T and adds at most N^2 additional I/Os, which any attention algorithm must use whenever $M < d^2$. We give the reduction in terms of the computational graph below.

Suppose for contradiction there is an algorithm $\mathcal{A}$ computing attention with I/O complexity $o\left(\frac{N^2 d}{\sqrt{M}}\right)$. Consider $\mathcal{A}$ as a complete calculation on the computational graph described in Figure 3. Since $\mathcal{A}$ is a complete calculation and the set of input and output vertices are disjoint, every single vertex in the graph must have a pebble on it at some configuration in the calculation. Consider then the algorithm $\mathcal{B}$ which executes $\mathcal{A}$ with the following modifications:

1. Whenever a blue pebble is deleted from a vertex in QK^T, do not delete.
2. Whenever a red pebble is placed on a vertex in QK^T for the first time, place also a blue pebble on this vertex.

The two properties guarantee that $\mathcal{B}$ will have at least the pebbles that $\mathcal{A}$ has, while any additional pebbles must be blue, so that $\mathcal{B}$ respects the constraint on the overall number of red pebbles at any given configuration. In particular, $\mathcal{B}$ is a valid calculation that computes QK^T. We now analyze the I/O complexity of $\mathcal{B}$. If $Q_{\mathcal{A}}$ denotes the I/O complexity of $\mathcal{A}$, the additional writes due to the second rule imply an overall I/O complexity of,

$$Q_{\mathcal{A}} + N^2 = o\left(\frac{N^2 d}{\sqrt{M}}\right)$$

Since $\mathcal{B}$ computes QK^T, this contradicts Lemma A.2. □

B. I/O Complexity of Attention with Fast Matrix Multiplication - Omitted Proofs

In this section, we provide the omitted proofs of Section 4.

Lemma 4.1. *[Theorem 3 of (Pagh & Silvestri, 2014)] Suppose $\mathcal{A}'$ is an execution of algorithm $\mathcal{A}$ on a machine with cache of size M. The execution $\mathcal{A}'$ can be split into T epochs of at most M I/O operations, such that in each epoch the algorithm $\mathcal{A}$ has access to a cache of size at most $2M$ and no I/O operations. Furthermore, the I/O complexity of $\mathcal{A}'$ is at least $(T-1)M$.*

Proof. Let $\mathcal{A}$ be any algorithm computing exact attention and $\mathcal{A}'$ an arbitrary execution of $\mathcal{A}$ on machine with a cache size of M bits. Note that given $\mathcal{A}'$ all the decisions made by algorithm $\mathcal{A}$ are already taken.

We proceed to simulate the execution $\mathcal{A}'$ on a machine with cache size of $2M$ so that the computation is split into epochs and I/O operations are performed only at the start and end of each epoch. Split the cache into two pieces, one block of size M to simulate the cache of $\mathcal{A}$ and one block as a buffer for I/O operations. At the start of the epoch, the simulation considers all of the M next I/O operations, performs all read I/Os by filling the buffer. During the epoch, any I/O operation is simulated by writing data between the two blocks of cache. Finally, at the end of the epoch, the simulation takes all written entries in the buffer and writes them to memory. In particular, in each epoch, no I/O operations are performed so that the algorithm only has access to only a cache of size $2M$ with no I/O.

Finally, in every epoch except for the last, M I/O operations are performed, so the I/O complexity of the execution $\mathcal{A}'$ is at least $(T-1)M$. □

Next, we give the simple lower bound of Lemma 4.5.

Lemma 4.5. *Let $B \geq 0$. Then, the one-way communication complexity of* $\mathbf{MatrixEntryCompression}_B$ *is at least $\sqrt{B} \log q$.*

Proof. Let $I \subset [N]^2$ denote any set of indices of the computed entries of QK^T where $|I| = B$. Define R_I to be the distinct row indices in I and C_I to be the distinct column indices in I so that $I \subset R_I \times C_I$.

Without loss of generality, assume $|R_I| \geq |C_I|$. We claim there are at least $q^{|R_I|}$ distinct values in the entries of QK^T indexed by I. In particular, let Q, K both be matrices with non-zero values only in the first column. In K, we set every entry in the first column to 1. In this case, each row of QK^T will be the same, so we can assume the B entries are $|R_I|$ entries in a column of QK^T. Since we can arbitrarily set the entries of Q indexed by R_I, we can obtain $q^{|R_I|}$ different outputs. Since there are at least $q^{|R_I|}$ outputs, the message length M must be at least $|R_I| \log q = \max(|R_I|, |C_I|) \log q$ in order to unambiguously determine the correct output. Then,

$$B \leq R_I C_I \leq \max(R_I, C_I)^2 \leq \left(\frac{M}{\log q}\right)^2$$

Thus, $M \geq \sqrt{B} \log q$. □

Next, we show that Vandermonde matrices satisfy the required linear independence constraints.

Lemma 4.7. *There is a $N \times d$ Vandermonde matrix with entries in a finite field $\mathbb{F}_q$ of size $q > N$ where every subset of d rows is linearly independent.*

Proof. Consider the $N \times d$ Vandermonde matrix,

$$V = \begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \dots & \alpha_1^{d-1} \\ 1 & \alpha_2 & \alpha_2^2 & \dots & \alpha_2^{d-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha_N & \alpha_N^2 & \dots & \alpha_N^{d-1} \end{pmatrix}$$

where $\alpha_1, \alpha_2, \dots, \alpha_N$ are distinct elements in $\mathbb{F}_q$, since we choose $q > N$. Then, for any subset of d rows, the determinant of this sub-matrix is,

$$0 \neq \prod_{1 \leq i < j \leq d} (\alpha_{k_i} - \alpha_{k_j})$$

where $k_1, \dots, k_d$ are the indices of the d rows. Since the determinant of this matrix is non-zero, the rows are linearly independent. □

B.1. I/O Complexity via Binary Matrix Compression

In this section, we provide the omitted details for I/O complexity of attention on binary matrices. First, we state the following standard lemma relating distance of linear codes to linear independence in the parity check matrix.

Lemma B.1. *A linear code has distance d, if and only if any $(d-1)$ columns of the parity check matrix is linearly independent and there exist d columns that are linearly dependent.*

Proof. Consider a code $\mathcal{C}$ of length n, dimension k, and distance d. Suppose there is a set of $(d-1)$ linearly dependent columns. Then, there is a vector x with $\mathtt{wt}(x) \leq d-1$ such that $Hx = 0$, contradicting the minimum distance d of $\mathcal{C}$. Since the distance of the code is d, there exists a vector x such that $Hx = 0$ and $\mathtt{wt}(x) = d$. In particular, there exists a subset of d independent columns.

To prove the converse, note that the conditions on H imply there exists a codeword of weight d and no codeword of weight less than d, so that the minimum distance of the code is exactly d. □

Next, we use the fact that binary BCH codes are optimal high rate codes. Recall that a code with parity check matrix H of dimension $d \times N$ has dimension at least $N - d$.

Lemma B.2. *(Hocquenghem, 1959; Bose & Ray-Chaudhuri, 1960) For a length $N = 2^m - 1$ and a distance s, there exists a code $\mathrm{BCH}[N, s]$ with dimension at least $N - \left\lceil \frac{s-1}{2} \right\rceil \log(N+1)$.*

We provide the definition of BCH codes. Recall an element $\alpha \in \mathbb{F}$ in a finite field is a *primitive* element if it generates the multiplicative group $\mathbb{F}^*$.

Definition B.3 ((Hocquenghem, 1959; Bose & Ray-Chaudhuri, 1960))**.** For length $N = 2^m - 1$, distance s, and primitive element $\alpha \in \mathbb{F}_{2^m}^*$, the binary BCH code is defined,

$$\mathrm{BCH}[N, s] = \{(c_0, \ldots, c_{N-1}) \text{ s.t. } c(\alpha) = \ldots = c(\alpha^{s-1}) = 0\}$$

where $c(X) = c_0 + c_1 X + \ldots + c_{N-1} X^{N-1}$.

The proof of Lemma B.2 is a standard exercise.

Proof of Lemma B.2. To show a lower bound on the dimension of the code, we argue that the parity check matrix H does not have too many rows. We begin with a weaker bound of $N - (s-1)\log(N+1)$. In particular, we show that each constraint c can be written as $m = \log(N+1)$ linear constraints.

We choose a basis $\beta = \{\beta_1 = 1, \beta_2, \ldots, \beta_m\}$ of $\mathbb{F}_2^m$ as a vector space. For any $x \in \mathbb{F}_2^m$, consider the linear map $x \mapsto \alpha x$ which can be written as $x \mapsto M_\alpha x$ for some matrix $M_\alpha \in \mathbb{F}_2^{m \times m}$ where x is represented in the above basis. Then, the constraint $c(\alpha) = 0$ can be viewed as,

$$\begin{pmatrix} c_0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + M_\alpha \begin{pmatrix} c_1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \ldots + M_{\alpha^{N-1}} \begin{pmatrix} c_{N-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Thus, each constraint $c(\alpha_i) = 0$ can be viewed as $\log(N+1)$ linear constraints. Since there are $s-1$ such constraints, this ensures the parity check matrix has dimension $(s-1)\log(N+1) \times N$.

We now prove the improved bound. This follows from the fact that $c(\gamma) = 0$ if and only if $c(\gamma^2) = 0$. In particular, $\left\lfloor \frac{s-1}{2} \right\rfloor$ constraints are redundant, leaving only $\left\lceil \frac{s-1}{2} \right\rceil$ relevant constraints.

It remains to show $c(\gamma^2) = 0$ if and only if $c(\gamma) = 0$. Note that $c(\gamma) = 0$ if and only if $c(\gamma)^2 = 0$. Furthermore, for $\alpha, \beta \in \mathbb{F}_2^m$, $(\alpha + \beta)^2 = \alpha^2 + \beta^2$. Then,

$$\begin{aligned} 0 = c(\gamma) &= c(\gamma)^2 \\ &= c_0^2 + (c_1\gamma)^2 + \ldots + (c_{N-1}\gamma^{N-1})^2 \\ &= c_0 + c_1\gamma^2 + \ldots c_{N-1}\gamma^{2(N-1)} = c(\gamma^2) \end{aligned}$$

where we have used $c_i = c_i^2$ for all coefficients $c_i \in \mathbb{F}_2$. □

We now prove the construction of the desired matrix K satisfying strong linear independence constraints.

Lemma 4.9. *There exists a matrix $K \in \{0,1\}^{N \times d}$ such that every set of $\frac{2d}{\log(N+1)} - 1$ rows is linearly independent.*

Proof. We assume without loss of generality that $N = 2^m - 1$ for some m. If not, we at most double N by choosing the minimum m such that $2^m - 1 \geq N$ and take any N-row sub-matrix.

From the construction of the code $\mathrm{BCH}[N, s]$, we observe that it has a parity check matrix H of dimension $\left\lceil \frac{s-1}{2} \right\rceil \log(N+1) \times N$. Since the code has distance s, from Lemma 4.9, any set of $s-1$ rows is linearly independent. In particular, if $d = \left\lceil \frac{s-1}{2} \right\rceil \log(N+1)$, we have,

$$s \geq \frac{2d}{\log(N+1)}$$

giving the desired bound on $s - 1$. Thus, we choose $K = H^T$. □

Given this matrix construction, we prove an analogue of Lemma 4.6 for the binary input case.

Lemma 4.10. *Suppose Q, K are binary $N \times d$ matrices. Then, the one-way communication complexity of* $\mathbf{MatrixEntryCompression}_B$ *is at least $\Omega\left(\min\left(\frac{d\sqrt{B}}{\log N}, B\right)\right)$.*

Proof. The proof follows Lemma 4.6 closely, so we only point out the necessary modifications. Again let M denote the maximum message length in the communication protocol and $I = [N]^2$ denote the indices of QK^T computed with $B = |I|$. Define $R_I, C_I, \{R_i\}_{i=1}^N, \{C_j\}_{j=1}^N$ as in Lemma 4.6.

We modify $L_R = \{i \text{ s.t. } |R_i| \geq \frac{2d}{\log(N+1)} - 1\}$ and $L_C = \{j \text{ s.t. } |C_j| \geq \frac{2d}{\log(N+1)} - 1\}$ and define S_R and S_C as before.

We again consider first the case $\max(|L_R|, |L_C|) \geq \sqrt{B/2}$ and assume $|L_R| \geq \sqrt{B/2}$. Instead of the Vandermonde matrix, we fix K to be the matrix guaranteed by Lemma 4.9. In particular, every subset of $\frac{2d}{\log(N+1)} - 1$ columns in K^T is linearly independent.

Following an analogous argument as Lemma 4.6, we obtain the following lower bound on the message length of the communication protocol in order to unambiguously specify B entries,

$$\begin{aligned} M &\geq \sum_{i \in R_I} \min\left(|R_i|, \frac{2d}{\log(N+1)} - 1\right) \\ &\geq \left(\frac{2d}{\log(N+1)} - 1\right)|L_R| + |S_R| \\ &= \Omega\left(\frac{d\sqrt{B}}{\log N}\right) \end{aligned} \tag{2}$$

where the final inequality follows from our assumption on L_R.

Finally, we consider the case $\max(|L_R|, |L_C|) < \sqrt{B/2}$. Again following similar arguments as Lemma 4.6 and Equation 2, we have,

$$M \geq \max(|S_R|, |S_C|) \geq \frac{B}{4}$$

so that M is at least the minimum of the two bounds. □

Finally, we show a lower bound for any algorithm computing attention on binary input matrices.

Theorem 4.11. *Suppose $Q, K \in \{0,1\}^{N \times d}$. The I/O complexity of attention (with any matrix multiplication algorithm) is $\Omega\left(\min\left(\frac{N^2d^2}{M \log^2 N}, N^2\right)\right)$.*

Proof. Following similar arguments as Theorem 4.8, we obtain,

$$\Omega\left(\min\left(\frac{d\sqrt{B^*}}{\log N}, B^*\right)\right) \leq 2M$$

where B^* is as defined in Theorem 4.4. In particular, the maximum number of entries of QK^T computed in any epoch is at most,

$$B^* = O\left(\max\left(\frac{M^2 \log^2 N}{d^2}, M\right)\right)$$

which gives the I/O complexity lower bound,

$$\Omega\left(\min\left(\frac{N^2d^2}{M \log^2 N}, N^2\right)\right)$$

as desired. □

B.2. Small Cache: Attention and Matrix Multiplication Equivalence

Finally, we prove the equivalence of Attention and Matrix Multiplication in the small cache setting.

Lemma 4.12. *For all M,*

$$Q_{Att}(M) = O\left(Q_{\mathcal{M}(N,d,N)}(M) + Q_{\mathcal{M}(N,N,d)}(M)\right)$$

Proof. First, we use the given algorithm for rectangular matrix multiplication to compute QK^T with $Q_{\mathcal{M}(N,d,N)}$ I/O operations. Then, note that computing $\mathrm{softmax}(QK^T)$ can be done in $O(N^2)$ time and therefore $O(N^2)$ I/O complexity as each operation can be performed with $O(1)$ I/O operations. Finally, we use the given algorithm to compute $\mathrm{softmax}(QK^T)V$ with $Q_{\mathcal{M}(N,N,d)}$. Then, the overall I/O complexity is,

$$\begin{aligned} Q_{Att}(M) &= O\left(Q_{\mathcal{M}(N,d,N)} + N^2 + Q_{\mathcal{M}(N,N,d)}\right) \\ &= O\left(Q_{\mathcal{M}(N,d,N)} + Q_{\mathcal{M}(N,N,d)}\right) \end{aligned}$$

since the I/O complexity of both matrix products is at least N^2, as either the input or output has size N^2. □

Lemma 4.13. *Let $M \leq d^2$. Then, $Q_{Att}(M) = \Omega\left(Q_{\mathcal{M}(N,d,N)}(M)\right)$.*

Proof. First, consider any two input matrices Q, K^T. We simulate the attention algorithm with the following modification: whenever an entry $(QK^T)_{ij}$ is computed for the first time, we write this entry to memory. Our modified algorithm successfully computes the matrix product QK^T using at most $O\left(Q_{Att}(M) + N^2\right)$ I/O operations. From Theorem 4.8, we have that whenever $M \leq d^2$, $Q_{Att}(M) = \Omega(N^2)$. As a result, we compute Attention with $O\left(Q_{Att}(M)\right)$ I/O complexity. □

C. The Red-Blue Pebble Game

Inspired by the pebble game often used to model space-bounded computation, (Hong & Kung, 1981) develops the Red-Blue Pebble Game to model I/O complexity. As with the standard pebble game, the Red-Blue Pebble Game is played on a directed acyclic graph, where nodes represent computations and edges represent dependencies. Throughout the game, pebbles can be added to, removed from, or recolored between red and blue in the graph, where red pebbles represent data in cache and blue pebbles represent data in slow memory. Given an upper bound on the number of red pebbles (cache size), the goal is to minimize the number of pebble recolorings (I/O operations) over the course of a computation. The game is defined formally below.

Definition 2.1. [Red-Blue Pebble Game (Hong & Kung, 1981)] Let G be a directed acyclic graph with a set of *input* vertices containing all vertices with no parents, and a set of *output* vertices containing all vertices with no children. A *configuration* is a pair of (not necessarily disjoint) subsets of vertices, one containing all vertices with red pebbles, and the other containing all vertices with blue pebbles.

The *initial* (resp. *terminal*) configuration is one in which only input (resp. output) vertices contain pebbles, and all of these pebbles are blue. The rules of the red-blue pebble game are as follows:

R1 (Input) A red pebble may be placed on any vertex with a blue pebble.

R2 (Output) A blue pebble may be placed on any vertex with a red pebble.

R3 (Compute) A red pebble may be placed on a vertex if all its parents have red pebbles.

R4 (Delete) A pebble may be removed from any vertex.

A *transition* is an ordered pair of configurations where the second can be obtained from the first following one of the above rules. A *calculation* is a sequence of configurations, where each successive pair forms a transition. A *complete calculation* is a calculation that begins with the initial configuration and ends with the terminal configuration.

In typical instances, including our paper, the input and output vertices are disjoint. Furthermore, the input vertices are typically exactly those with no parents, while the output vertices are exactly those with no children. To model bounded cache, we assume that there are at most M red pebbles on the graph at any given time, while any number of blue pebbles can be placed on the graph.

The key idea is that any complete calculation implies an $2M$-partition of the computational graph G.

Definition 2.5. [M-partition (Hong & Kung, 1981)] Let G be a directed acyclic graph. A family of subsets $\{V_i\}_{i=1}^h$ is a *M-partition* of G if the following properties hold:

P1 (Partition) The sets V_i are disjoint and $V = \bigcup_{i=1}^{h} V_i$.

P2 (Dominator) For each V_i, there exists a dominator set D_i of size at most M.

P3 (Minimum) For each V_i, the set of minimum vertices M_i has size at most M.

P4 (Acyclic) There is no cyclic dependence among vertex sets $\{V_i\}_{i=1}^{h}$ (Definition C.1).

We give the missing definition of vertex subset dependence below.

Definition C.1 (Vertex Subset Dependence)**.** Let $S, T \subset V$ be disjoint subsets. T *depends* on S if there exists an edge $(s, t) \in E$ with $s \in S, t \in T$.

Specifically, by showing a lower bound on the size of any $2M$-partition, we may immediately obtain a lower bound on the I/O complexity of the algorithm represented by G.

Lemma C.2 (Theorem 3.1 of (Hong & Kung, 1981))**.** *Let G be a directed acyclic graph. Any complete calculation of the Red-Blue Pebble Game on G, using at most M red pebbles, is associated with a $2M$-partition of G such that,*

$$M \cdot h \geq Q(G, M) \geq M \cdot (h - 1)$$

where h is the number of vertex subsets in the $2M$-partition.

Roughly speaking, a M-partition partitions the vertex set such that each part can be computed completely using only $2M$ I/O operations. In particular, every node in a single part V' can be reached by placing red pebbles at the dominator vertices. This can be thought of as the initial state of the cache at some point, and without any further reads from memory, every node on V' can be computed. Then, after the computation, the values at the minimum vertices can be written to memory. This corresponds to at most M write operations.