Fast attention mechanisms: a tale of parallelism

Jingwen Liu
Columbia University
New York, NY
jingwenliu@cs.columbia.edu

Hantao Yu
Columbia University
New York, NY
hantao.yu@columbia.edu

Clayton Sanford
Google Research
San Francisco, CA
chsanford@google.com

Alexandr Andoni
Columbia University
New York, NY
andoni@cs.columbia.edu

Daniel Hsu
Columbia University
New York, NY
djhsu@cs.columbia.edu

Abstract

Transformers have the representational capacity to simulate Massively Parallel Computation (MPC) algorithms, but they suffer from quadratic time complexity, which severely limits their scalability. We introduce an efficient attention mechanism called Approximate Nearest Neighbor Attention (ANNA) with sub-quadratic time complexity. We prove that ANNA-transformers (1) retain the expressive power previously established for standard attention in terms of matching the capabilities of MPC algorithms, and (2) can solve key reasoning tasks such as Match2 and k-hop with near-optimal depth. Using the MPC framework, we further prove that constant-depth ANNA-transformers can simulate constant-depth low-rank transformers, thereby providing a unified way to reason about a broad class of efficient attention approximations.

1 Introduction

The transformer [57] has become the dominant neural architecture in deep learning due to its ability to select and compose complex information structures from large inputs [1, 19, 24], which in turn enables capabilities such as "in-context learning" [13] that are crucial for downstream applications. At the core of transformers is the attention mechanism, which leverages parallelism to gain important advantages over non-parallel architectures (e.g., recurrent nets), including training stability [46] and representational power [11, 31, 41, 45, 52, 55, 60]. One recently highlighted advantage comes from a coarse theoretical relationship between transformers and the Massively Parallel Computation (MPC) model [33] that captures the power of large-scale distributed computing frameworks like MapReduce [17]: efficient MPC algorithms can be simulated by small-size transformers, and vice versa [53, 55]. This correspondence suggests that a broad class of computational tasks can be efficiently solved by transformers.

Despite these advantages, transformers suffer from a key limitation: the quadratic time complexity of (standard) attention with respect to input size, which is likely to be unavoidable in the worst-case [2, 3, 34]. To address this limitation, a growing body of work proposes alternatives to the attention mechanism that are more computationally efficient (e.g., sub-quadratic time). These alternatives employ a variety of techniques, ranging from low-rank approximation [8, 14, 16, 32, 58] to efficient nearest neighbor search [27, 35, 51, 64]. Although many of these techniques show promising empirical performance, it is unclear whether they preserve the representational advantages of attention.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

In this work, we study a particular efficient (sub-quadratic time) attention mechanism, called Approximate Nearest Neighbor Attention (ANNA), and we prove that ANNA retains key representational advantages of standard attention over non-parallel architectures. We additionally prove that ANNA-transformers can simulate a class of attention mechanisms based on low-rank approximation. In doing so, we provide a unified way to reason about low-rank and nearest neighbor approaches to efficient attention.

1.1 Standard transformers and MPC

Prior work [53, 55] established a coarse relationship between standard transformers and MPC by proving the following (for any constant $\delta \in (0, 1)$):

1. For any MPC algorithm with inputs of size N that uses R rounds (of computation and communication), $O(N)$ machines, and $O(N^\varepsilon)$ words of local memory per machine (for some constant $\varepsilon \in (0, 1)$), there is an equivalent transformer of width $O(N^{\varepsilon+\delta})$ and depth $O(R)$. (By width, we mean number of heads per layer times the embedding dimension.)
2. For any transformer operating on inputs of size N that has L layers and width $O(N^\varepsilon)$ (for some constant $\varepsilon \in (0, 1)$), there is an equivalent MPC algorithm that uses $O(L)$ rounds, $O(N^2)$ machines, and $O(N^{\varepsilon+\delta})$ words of local memory per machine.

Notice that the "loss" in the number of rounds or depth is only a constant factor, and the "loss" in the local memory size or width is only a $O(N^\delta)$ factor (where $\delta > 0$ can be arbitrarily small). Consequently, these results were sufficient to give new results about transformer representational power (e.g., for various graph reasoning tasks [53, 55]), and also give a (conditional) logarithmic-depth lower bound for transformers that solve a multi-hop reasoning problem called k-hop [55].

However, there is a important gap in the MPC algorithm for simulating a transformer (Item 2 above): *it may use up to N^2 machines*. This gap suggests the possibility that transformers are strictly more powerful than MPC algorithms, so the relationship established in prior work is very coarse. Moreover, this gap is likely to be inevitable. Indeed, when the local computation in an MPC algorithm is fast (say, polynomial-time), and the local memory size is N^ε for small $\varepsilon \in (0, 1)$, then a single round of MPC on M machines can be simulated on a sequential machine in roughly $MN^{O(\varepsilon)}$ time. However, evaluation of attention over N inputs is believed to require at least $N^{2-\delta}$ time (for every constant $\delta > 0$) on a sequential machine [2, 3, 34]. So any such MPC algorithm that simulates attention in $\tilde{O}(1)$ rounds should use $N^{2-\delta-O(\varepsilon)}$ machines.

This gap naturally leads us to the following question: Is there an alternative attention mechanism that more tightly captures the computational power of MPC?

1.2 Our contributions

We prove that ANNA-transformers more sharply capture the power of MPC algorithms than standard transformers do, and in particular avoid the aforementioned gap in prior works' characterization of transformers using MPC [53, 55]. The core ANNA mechanism is designed to perform c-approximate nearest neighbor search [30, 38] for a given set of N "queries" against a database of N "key-value" pairs. Here $c > 1$ is a parameter of the ANNA mechanism that, for the purpose of this present informal description, should be thought of as being at least a large positive constant (e.g., $c \geq 10$). Throughout, N denotes the input size.

Theorem 1.1 (Informal version of Theorems 4.1 and 4.2)**.** *Fix any constants $\varepsilon, \delta \in (0, 1)$ and ANNA parameter $c > 1$. For any R-round MPC algorithm with N^ε words of local memory, there is an equivalent $O(R)$-layer ANNA-transformer of width $O(N^{\varepsilon+\delta})$. For any L-layer ANNA-transformer with width N^ε, there is an equivalent $O(L)$-round MPC algorithm that uses $N^{\varepsilon+\delta}$ words of local memory and $N^{1-\delta+O(1/c^2)}$ machines.*

Observe that Theorem 1.1 essentially mirrors Items 1 and 2 in Section 1.1 except for the resources needed by the MPC algorithm in Item 2: the local memory stays the same, but the number of machines required can be strongly sub-quadratic (and in fact, close to linear) in N. This gives a positive answer to the question at the end of Section 1.1.

We also study efficient attention mechanisms based on low-rank approximation, and by leveraging Theorem 1.1, we prove that ANNA-transformers have at least the same representational power.

Theorem 1.2 (Informal version of Theorem 4.4)**.** *For any L-layer* low-rank attention *transformer, there is an equivalent $O(L)$-layer ANNA-transformer (of comparable width).*

Finally, we illustrate the power of ANNA-transformers on two concrete reasoning tasks (Match2 [52] and k-hop induction heads [12, 55]). We give theoretical constructions of ANNA transformers for these tasks that nearly match the efficiency achievable by standard transformers (Theorems 5.2 and 5.6), and we also show empirically that ANNA-transformers can be trained to approximately solve these tasks (Section 5.3).

1.3 Other related works

The representational power of standard attention is relatively well understood from a variety of perspectives. This includes: universal approximation properties in the large depth/width limit [22, 44, 59, 63], ability to recognize formal languages [10, 25, 41, 56, 62], computational bounds in terms of circuit classes [28, 39, 41, 43] and other parallel computation models (discussed below) [53, 55], and bounds for specific compositional tasks [12, 15, 36, 41, 49, 54]. In general, these and other prior works do not consider the representational power of transformers based on efficient (sub-quadratic time) alternatives to attention. The exceptions are works that give lower bounds for low-rank attention and sparse attention [55, 61], sequential architectures [11, 31, 45, 52, 60], more generally, any mechanism that can be evaluated in sub-quadratic time [2, 3, 34]. What is missing in these prior works, however, is a characterization (and, in particular, upper bounds) for an efficient architecture.

Massively Parallel Computation (MPC) [5, 9, 20, 23, 29, 33] is a model of parallel computing intended to capture the power of MapReduce [17] and other large-scale distributed computing frameworks. Many works, including those that originally helped to define the MPC model, gave efficient algorithms for a variety of basic data and graph processing tasks. A connection between MPC and circuit models was given by [50] and used to formalize a barrier on proving lower bounds for certain graph problems (e.g., connectivity) in MPC. Nevertheless, certain conjectured lower bounds in MPC are widely believed (e.g., the 1-vs-2 cycle conjecture [29]) and have been used to establish conditional lower bounds for other problems [21]. The same conjectured lower bounds have also been used to establish depth lower bounds for transformers via the aforementioned relationship between MPC and transformers [53, 55].

Our main result for ANNA-transformers has an analogue in the context of message-passing graph neural networks (GNNs). The computational power of such GNNs is characterized by the CONGEST model of distributed computing [48] (a refinement of LOCAL [7, 40, 47]). This fact was established by [42] and in turn used to give upper- and lower-bounds on the size of GNNs needed to solve various graph problems (e.g., subgraph detection). The GNN/CONGEST equivalence is almost immediate, since the communication network is fixed by the input graph in both models, and the model definitions are semantically and syntactically similar. In contrast, while the layered representations of transformers are reminiscent of the alternation between communication and computation rounds in MPC, the "communication patterns" themselves are dynamic, and this dynamism greatly complicates the simulation of MPC algorithms by either standard transformers or ANNA-transformers.

The ANNA mechanism that we study is inspired by many prior efficient attention mechanisms [e.g., 27, 35, 64] based on locality-sensitive hashing [30], which is one of the key techniques for nearest neighbor search. Some of these mechanisms have guarantees about the quality of approximation under structural assumptions on the attention matrix they are meant to approximate. The motivation of our analysis is largely orthogonal: we instead seek to characterize the representational power of ANNA-transformers in terms of other well-understood models of parallel computation. We further compare the capabilities of LSH-based efficient attention to other sub-quadratic alternatives, including those based on low-rank approximations of self-attention matrices [16, 32, 58].

2 Preliminaries

2.1 Standard attention and transformers

We first define the (standard) attention mechanism and transformers.

Definition 2.1 (Attention)**.** *A* (standard) attention head $\text{Attn}_{Q,K,V}$ *is specified by query, key, value embedding functions* $Q, K, V : \mathbb{R}^d \to \mathbb{R}^m$*. On input* $X \in \mathbb{R}^{N\times d}$*, it computes*

$$\text{Attn}_{Q,K,V}(X) := \text{softmax}(Q(X)K(X)^\intercal)V(X) \in \mathbb{R}^{N\times m},$$

where Q*,* K*,* V*, and* softmax *are applied row-wise. We say* N *is the* context length*, and* m *is the* embedding dimension*. The rows of* $Q(X)$ *(resp.,* $K(X)$*,* $V(X)$*) are the* queries *(resp.,* keys*,* values*). If* $q_i = Q(X)_i$*,* $k_j = K(X)_j$*, and* $v_j = V(X)_j$*, then the* i*-th row of* $\text{Attn}_{Q,K,V}(X)$ *is*

$$\text{Attn}_{Q,K,V}(X)_i = \sum_{j=1}^{N} w_{i,j}v_j, \quad \textit{where} \quad w_{i,j} = \frac{\exp(\langle q_i, k_j\rangle)}{\sum_{j'=1}^{N}\exp(\langle q_i, k_{j'}\rangle)}.$$

An H*-headed* attention layer $f\colon \mathbb{R}^{N\times d} \to \mathbb{R}^{N\times d}$ *consists of attention heads* $(\text{Attn}_{Q_h,K_h,V_h})_{h=1}^{H}$ *and* $m \times d$ *matrices* $(W_h)_{h=1}^{H}$*; it computes* $f(X) := \sum_{h=1}^{H}\text{Attn}_{Q_h,K_h,V_h}(X)W_h$*.*

Definition 2.2 (Transformer)**.** *An* L*-layer* transformer $\mathcal{T}$ *is specified by attention layers* $f_1, \ldots, f_L : \mathbb{R}^{N\times d} \to \mathbb{R}^{N\times d}$ *and an output function* $\psi : \mathbb{R}^d \to \mathbb{R}^d$*. Given input* $X \in \mathbb{R}^{N\times d}$*, define*

$$X^{(0)} := X \quad \textit{and} \quad X^{(\ell)} := f_\ell(X^{(\ell-1)}) \quad \textit{for } \ell = 1, \ldots, L.$$

The output of $\mathcal{T}$ *on input* X *is* $\psi(X^{(L)})$ *with* ψ *being applied row-wise.*

In this paper, we consider the context length N (the number of input tokens) as the principal scaling parameter. This reflects the modern paradigm of long-context LLMs, where the context-length can exceed 10^6 [19], enabling book-length textual inputs. Consequently, we typically want the size parameters m, H, and L to be sub-linear in N (and L ideally constant). The $O(N^2)$ runtime of attention therefore remains the main bottleneck of the transformer architecture.

Following [55], we allow the element-wise operations (Q_h, K_h, V_h, ψ) to be arbitrary functions [1] (limited only by bit-precision; see [55, Appendix A.1]). Therefore, we can view a transformer as a computational model that alternates between arbitrary per-token computation, and communication between tokens. This motivates a connection to massively parallel computation, defined next.

2.2 Massively Parallel Computation

The Massively Parallel Computation (MPC) framework [29] models computation on large inputs by a distributed computing system that alternates between rounds of local computation and rounds of restricted all-to-all communication. We formally state the definition of MPC protocols (with sublinear memory) as follows.

Definition 2.3 (MPC)**.** *For constants* $\gamma, \varepsilon > 0$*, an* R*-round* (γ, ε)*-MPC protocol specifies the following computation on inputs of* N *words (where a word is* $p = \Theta(\log N)$ *bits, represented by an element of* $\mathbb{Z}_{2^p}$*) by* $P = \Theta(N^{1+\gamma-\varepsilon})$ *machines, each with local memory* $s = O(N^\varepsilon)$ *words:*

1. *Initially, the input is arbitrarily distributed across the first* $\lceil \frac{N}{s} \rceil$ *machines.*
2. *In each round, each machine prepares, as an arbitrary function of its local memory, messages to send to other machines. The total size of messages prepared by any machine is at most* s *words.*
3. *At the end of each round, the messages are placed in the local memory of the intended recipients. The protocol ensures that the messages received by any machine has total size at most* s *words.*
4. *After the* R*-th round, the output is stored in the memory of the first* $\lceil \frac{N}{s} \rceil$ *machines.*

We say an MPC protocol π *computes a function* $f : \mathbb{Z}_{2^p}^N \to \mathbb{Z}_{2^p}^N$*, if for any* $X \in \mathbb{Z}_{2^p}^N$*,* $\pi(X) = f(X)$*, where* $\pi(X)$ *is the output of* π *with the input* X*.*

The primary measure of complexity considered in this paper is the number of rounds R. The round complexity of numerous classical algorithmic problems is well understood. For example, there are simple MPC protocols for graph connectivity ($O(\log N)$ rounds) and sorting ($O(1)$ rounds) [20].

[1] Such assumptions are necessary for establishing the equivalence with the MPC model, since the MPC model allows arbitrary computation on the local memory for each machine. That said, many concrete MPC algorithms do have a simple local algorithm which can be simulated by a small MLP.

3 Approximate Nearest Neighbor Attention

In this section, we introduce Approximate Nearest Neighbor Attention (ANNA), an attention mechanism inspired by the approximate nearest neighbor (ANN) search problem. We first outline the approximate nearest neighbor search problem and present locality-sensitive hashing (LSH), a core technique for ANN (Section 3.1). We then formally define ANNA and provide a sub-quadratic time algorithm based on LSH for computing ANNA with theoretical guarantees (Section 3.2).

3.1 Approximate nearest neighbor and locality sensitive hashing

We first define the Approximate Nearest Neighbor (ANN) search problem.

Definition 3.1 (ANN search problem [30, 38])**.** *Given a dataset D of N points lying in a metric space Y and parameters $c, r > 0$, build a data structure that, given a query $q \in Y$ within distance at most r from D, returns any point in D that is within distance cr from q.*

In the modern machine learning setting, we want to develop fast algorithms for ANN search in the high-dimensional metric space with sub-linear query time. A well-known tool that achieves this runtime and provable approximation guarantees is locality sensitive hashing (LSH). For simplicity, we assume the metric space is m-dimensional Euclidean space.

Definition 3.2 (Locality Sensitive Hashing [30])**.** *Fix a parameter $r > 0$, an approximation factor $c > 1$ and a set U. Then a family $\mathcal{H}$ of hash functions $h : \mathbb{R}^m \to U$ is (r, cr, p_1, p_2)-sensitive if the following holds for any $x, y \in \mathbb{R}^m$:*

- *if $\|x - y\| \le r$, then $\Pr_{h\in\mathcal{H}}[h(x) = h(y)] \ge p_1$, and*
- *if $\|x - y\| > cr$, then $\Pr_{h\in\mathcal{H}}[h(x) = h(y)] \le p_2$.*

The family $\mathcal{H}$ is called an LSH family with quality $\rho = \frac{\log(1/p_1)}{\log(1/p_2)}$.

A typical LSH-based algorithm can solve the ANN search problem with space $O(N^{1+\rho})$ and query time $O(N^\rho)$, and ρ can be as small as $1/c^2$ [4].

3.2 Transformer based on Approximate Nearest Neighbor Attention

We first define a family of models where only tokens with sufficiently "neighborly" queries and keys attend to one another and then provide an efficient implementation of a subset of this class. ANNA attention units treat attention query vectors as queries to approximate nearest neighbor search and key vectors as data points. ANNA retrieves and weights value vectors according to approximate nearest neighbor thresholds. The following definition formalizes this family of models.

Definition 3.3 (ANN Attention)**.** *An* Approximate Nearest Neighbor Attention (ANNA) *mechanism* $\text{ANNA}_{Q,K,V}$ *with query, key, and value embedding functions $Q, K, V : \mathbb{R}^d \to \mathbb{R}^m$ and (non-negative) parameters r, c, ℓ, η, is a (possibly randomized) mechanism that performs the following computation on an input $X \in \mathbb{R}^{N\times d}$:*

$$\text{ANNA}_{Q,K,V}(X)_i := \sum_{j=1}^{N} w_{i,j} v_j \quad \textit{for all } i \in [N],$$

for some non-negative weights $w_{i,j} \ge 0$ with $\sum_j w_{i,j} = 1$ that satisfy the following. With probability at least $1 - \eta$, for all $i \in [N]$,

- $w_{i,j} > 0 \Rightarrow k_j \in \mathcal{N}(q_i, cr)$
- $k_j \in \mathcal{N}(q_i, r) \Rightarrow w_{i,j} \ge \frac{1}{(|\mathcal{N}(q_i,cr)|-1)\ell+1}$

where $q_i := Q(X)_i$, $k_i := K(X)_i$, $v_i := V(X)_i$, and $\mathcal{N}(q, t) := \{k \in \{k_j\}_{j=1}^N : \|q - k\| \le t\}$.

We define ANNA layers *and* ANNA transformers *in a completely analogous fashion as (standard) attention layers and transformers are defined (Definitions 2.1 and 2.2).*

Algorithm 1 ANNA implementation with LSH family $\mathcal{H}$, ℓ hash tables, and z hash functions/table

Input: Input $X \in \mathbb{R}^{N \times d}$
Output: ANNA output for each of the query.

Let $q_i = Q(X)_i$, $k_i = K(X)_i$, and $v_i = V(X)_i$ for all $i \in [N]$.
for $u = 1$ to ℓ **do** ▷ **Preprocessing phase**
    Sample z hash functions $h_{u,1}, h_{u,2}, \ldots, h_{u,z}$ i.i.d. from $\mathcal{H}$.
    Create empty hash table T_u indexed by hash codes (below).
    for each key-value pair (k_j, v_j) **do**
        Compute hash code $g_u(k_j) = (h_{u,1}(k_j), h_{u,2}(k_j), \ldots, h_{u,z}(k_j))$.
        if $T_u[g_u(k_j)]$ is empty, **then** $T_u[g_u(k_j)] := (v_j, 1)$ **else** $T_u[g_u(k_j)]$ += $(v_j, 1)$.
Initialize a dictionary **attn** $\leftarrow \{(q_1, 0), (q_2, 0), \ldots, (q_N, 0)\}$
for each query q_i **do** ▷ **Query phase**
    $v_{\text{sum}} \leftarrow 0$; count $\leftarrow 0$
    for $u = 1$ to ℓ **do**
        Compute hash code $g_u(q_i) = (h_{u,1}(q_i), \ldots, h_{u,z}(q_i))$
        if $T_u[g_u(q_i)] = (v, a)$ is not empty, **then** v_{sum} += v and count += a
    attn$[q_i] \leftarrow v_{\text{sum}}$/count
return attn

The parameters r, c have the same semantics as in ANN search. The parameter ℓ captures how much "attention weight" is spread over keys that are not r-near neighbors of a query. The failure probability η allows for randomization, which is typical of ANN search algorithms like LSH.

The above definition represents a set of constraints that all ANNA units must satisfy rather than a specific algorithmic implementation. As a result, a wide variety of models satisfy the definition, including softmax attention with bounded query and key vectors, and *Exact-Match Attention (EMA)* where $w_{i,j} > 0$ if and only if $q_i = k_j$. Not all such models admit computationally efficient implementations. To identify sub-quadratic ANNA models, we present an LSH-based implementation of ANNA that computes satisfying weight vectors $w_{i,j}$ for specific choices of parameters r, c, ℓ, η.

We define a hash function family $G = \{g : p \in \mathbb{R}^d \mapsto (h_1(p), h_2(p), \ldots, h_z(p)) \in U^z \mid h_i \in \mathcal{H},\ \forall i \in [z]\}$ and sample ℓ hash functions, $g_1, \ldots, g_l$, from G independently and uniformly at random, giving ℓ hash tables. Each hash code corresponds to a hash bucket in the hash table and each hash bucket maintains a sum of v's and count of k's that falls into this bucket. We preprocess all the key, value pairs by storing them in the hash tables. For each $(k_i, v_i), i \in [N]$, compute the hash codes of k_i, $g_1(k_i), g_2(k_i), \ldots, g_\ell(k_i)$, and update the sum and count for the buckets corresponding to $g_1(k_i), g_2(k_i), \ldots, g_\ell(k_i)$ respectively. For each query $q_i, i \in [N]$, search and retrieve all the values and counts from $g_1(q_i), g_2(q_i), \ldots, g_\ell(q_i)$. Then compute the averaged value by summing up all the values in the ℓ buckets, divided by the sum of counts. See Algorithm 1 for the details.

Theorem 3.4 (LSH algorithm guarantee for ANNA)**.** *Fix $c > \sqrt{3}$, LSH family $\mathcal{H}$ that is (r, cr, p_1, p_2)-sensitive with quality $\rho < 1/3$, $\ell = \Theta(N^{3\rho} \log N)$, and $z = \Theta(\log_{1/p_2} N)$. Then Algorithm 1 (with $\mathcal{H}$, ℓ, and z) implements an ANNA mechanism with parameters r, c, ℓ and $\eta = O(1/N^{1-3\rho})$.*

We leave the full proof of Theorem 3.4 to Appendix A. The total runtime of Algorithm 1 is $O(mN^{1+3\rho} \log_{1/p_2} N)$, assuming sampling from the LSH family and evaluating a hash function requires $O(m)$ time and the numerical inputs to Algorithm 1 are specified with $p = \Theta(\log N)$ bits of precision. The total space used is $\tilde{O}(mN^{1+3\rho})$ bits.

Remark 3.5. *The memory complexity of Algorithm 1 can be further improved to $\tilde{O}(mN)$ bits with the same time complexity by storing only one hash table with each entry keeping track of the value for each query. See Algorithm 2 for the detailed implementation in Appendix A.*

Remark 3.6. *The weight $w_{i,j}$ depends on the number of hash collisions between q_i and k_j, and is typically a function of the distance $\Delta := \|q_i - k_j\|$. For example, if we use the random hyperplane LSH from [6], then $w_{i,j} \propto \exp(-\Delta^2 \log(m)/(4 - \Delta^2))$.*

In the remainder of this paper, our ANNA-based constructions are meant to refer to their efficient implementation by Algorithm 1 with a suitable choice of r and arbitrarily large c.

4 Efficient transformers and MPC

We prove a sharp equivalence between ANNA-transformer and MPC in the regime of sub-linear local memory and sub-quadratic number of machines (Sections 4.1 and 4.2). We also show that ANNA subsumes alternative low-rank sub-quadratic attention mechanisms. (Section 4.3).

4.1 ANNA-transformer can simulate MPC

The following theorem shows that any R-round MPC protocol with sub-linear local memory can be simulated by an ANNA-transformer with $O(R)$ layers and sub-linear number of heads and embedding dimension. The full proof of Theorem 4.1 is in Appendix B.

Theorem 4.1 (ANNA simulates MPC)**.** *Fix constants $0 < \varepsilon < \varepsilon' < 1$. For any deterministic R-round $(\varepsilon, \varepsilon)$-MPC protocol π, there exists an ANNA-transformer T with $L = O(R)$ layers, $H = O(N^{(\varepsilon'-\varepsilon)/4})$ heads per layer, and embedding dimension $m = O(N^{\varepsilon'})$, such that $T(\texttt{input}) = \pi(\texttt{input})$ for all $\texttt{input} \in \mathbb{Z}_{2^p}^N$.*

Proof sketch. In fact, we show that the special case of ANNA-transformer whose approximation factor $c \to \infty$ and $r = 0$ is already suffice to simulate MPC. In such case, ANNA for each query is equivalent to finding only the keys that exactly match the query; we call this Exact-Match Attention (EMA), and formally define it in Appendix B.

In our simulation, we treat each input token as the local machine, and all the local computation is handled by the element-wise functions Q, K, V. The bulk of the proof is to handle the message delivery between machines using EMA. By Proposition 24 of [53], we can assume that each machine only sends messages to at most $\alpha = O(N^{\delta})$ machines for some $\delta < \varepsilon$. We assign a unique positional encoding or identifier to each machine, and this encoding serves as a unique key to retrieve the message in each machine. The high level idea is to create a query for each machine and a key for each destination machine and the associated value is the embedding of the message sent to the destination machine in the protocol. Since each machine can send at most α messages to other machines, we create α EMA heads and each head is responsible for one outgoing message for all the N machines. Each machine retrieves the message sent to them by having a query in each head. Since the messages are averaged together, we use the same embedding mechanism from Lemma 3.2 of [55] to allow error correction in the element-wise operations. □

This gives us a sub-quadratic time reduction from MPC to ANNA-transformer: i.e., the communication process can be implemented in near linear time, whereas it is quadratic for standard attention. In addition, this ties ANNA-transformer in the existing MPC hierarchy [53]: any problem solvable by an R-round, $O(N^{\varepsilon})$-memory MPC protocol can be solved by $O(R)$-layer ANNA-transformer with $mH = O(N^{\varepsilon+\delta})$, for some $\delta > 0$. For example, following Theorem 3.1 of [26], $O(1)$-layer ANNA-transformer can solve 3-SUM with $mH = O(N^{1/2+\delta})$.

4.2 MPC can simulate ANNA-transformer

The following theorem (proved in Appendix C) shows that any L-layer ANNA-transformer (as implemented by Algorithm 1) can be simulated by a $O(L)$-round MPC protocol. Since Algorithm 1 is randomized, it uses a random seed to sample the hash functions from the LSH family. The simulation assumes access to the random seeds needed for all layers in the ANNA-transformer.

Theorem 4.2 (MPC simulates ANNA)**.** *Fix constants $0 < \varepsilon < \varepsilon' < 1$. For any L-layer ANNA-transformer T (as implemented by Algorithm 1) with $mH = O(N^{\varepsilon})$, there exists a $O(L/(\varepsilon' - \varepsilon))$-round MPC protocol π with local memory $s = O(N^{\varepsilon'})$ and $P = O(N^{1+\varepsilon-\varepsilon'+3/c^2})$ machines such that $\pi(\texttt{input}) = T(\texttt{input})$ for all $\texttt{input} \in \mathbb{Z}_{2^p}^N$.*

Observe that the number of machines used in the simulation of the ANNA-transformer can be strongly sub-quadratic (and in fact, close to linear when c is large). In contrast, the simulation of a standard transformer from [55] requires N^2 machines. As previously discussed (Section 1.2), this shows that ANNA-transformer more sharply characterizes efficient MPC protocols than standard transformers do. On the other hand, by Theorem 4.2, round-complexity lower bounds for MPC directly imply depth lower bounds for ANNA-transformers. This argument was used in [55] to establish (conditional)

depth lower bounds for standard transformers on problems such as graph connectivity and k-hop induction heads; these lower bounds also hold for ANNA-transformers.

4.3 ANNA-transformer can simulate low-rank transformers

As mentioned in Section 1, there are many proposals for efficient attention alternatives. In this section, we focus on the sub-quadratic alternatives based on low-rank approximations of the attention matrix. Specifically, we ask the following: *what problems are intrinsically easy for ANNA but hard for low-rank approximation attention, and vice versa?*

Definition 4.3 (Low-rank attention)**.** *A* low-rank attention *is specified by two feature maps $Q', K' : \mathbb{R}^d \to \mathbb{R}^r$ for some $r \ll N$ (with the intention of approximating* $\mathrm{softmax}(Q(X)K(X)^\intercal)$ *by $Q'(X)K'(X)^\intercal$). On input $X \in \mathbb{R}^{N\times d}$, it computes $Q'(X)K'(X)^\intercal V(X)$ by first computing $K'(X)^\intercal V(X) \in \mathbb{R}^{r\times m}$, and then left-multiplying by $Q'(X)$.*

Note that [55] gives a lower bound of any low-rank attention for the k-hop problem. Later in Section 5, we give a construction of $O(\log k)$-depth ANNA-transformer solving k-hop, and this directly gives a type of problem that is easy for ANNA but hard for low-rank attention. However, is there any problem that is easy for low-rank attention but hard for ANNA?

The following theorem answers the question by showing any L-layer low-rank attention-transformer can be simulated by a $O(L)$-layer ANNA-transformer. So, under the time and parameter-efficient regime (sub-linear rank and embedding dimension), low-rank attention-transformer is no stronger than ANNA-transformer.

Theorem 4.4 (ANNA simulates low-rank attention)**.** *For constants $0 < \varepsilon < \varepsilon' < 1$, any* low-rank attention *based transformer with depth L, rank r, embedding dimension m and $rm = O(N^\varepsilon)$ can be simulated by an ANNA-transformer with depth $O(L)$, number of heads $H = O(N^{(\varepsilon'-\varepsilon)/4})$ and embedding dimension $m = O(N^{\varepsilon'})$.*

We prove Theorem 4.4 by first using $O(L/(\varepsilon' - \varepsilon))$-round MPC to simulate L-layer low-rank transformer, and then Theorem 4.1 give us the simulation of L-layer low-rank transformer with ANNA-transformer through MPC. The full proof is given in Appendix D.

Other efficient attention mechanisms based on nearest neighbor search. Reformer [35] is another efficient attention based on LSH. In Reformer, the input tokens are sorted by their (scalar) hash values. Then, this sorted list is split into equal-sized chunks, each containing only $O(1)$-many tokens. Standard attention is applied within each chunk. We show that the expressive power of Reformer must come from the sorting operation: without sorting, the restriction of attention within each constant-size chunk prevents Reformer from even computing basic functions like "average" with $O(1)$ layers (regardless of the embedding dimension); details are given in Appendix E.

KDEformer [64] and HyperAttention [27] approximate softmax attention matrices as a sum of a sparse matrix and a low-rank matrix. They use LSH techniques to find sparse elements (i.e., heavy elements in the attention matrix) and low-rank attention for the remaining components. Theorem 4.4 indicates this low-rank part does not substantially increase the representational power.

5 ANNA-transformer for reasoning tasks

In this section, we study ANNA-transformer on two concrete reasoning tasks: Match2 [52] and k-hop [55]. These tasks are benchmarks for evaluating the reasoning capabilities of transformers, and they separate different neural architectures in terms of their representational strengths.

5.1 ANNA-transformer solves Match2

The Match2 task [52] measures the ability of a model to associate paired elements with one another. We show that a single ANNA mechanism can solve Match2.

Definition 5.1 (Match2)**.** *Given an input sequence $X = (x_1, \ldots, x_N) \in [M]^N$ for some $M \le \mathrm{poly}(N)$, the i-th output of* $\mathrm{Match2}(X)$ *is $\mathbb{1}\{\exists j \,.\, x_i + x_j = 0 \bmod M\}$ for all $i \in [N]$.*

Theorem 5.2. *For any $N, M = N^{O(1)}$, there exists an ANNA-transformer T with one layer, one attention head, and embedding dimension 1 such that $T(X) = \mathrm{Match2}(X)$ for all $X \in [M]^N$.*

5.2 ANNA-transformer solves k-hop

The induction heads (a.k.a. associative recall) task [18] is a reasoning task that predicts the next token by completing the most recent bigram. It has been identified as an important mechanism for the emergent "in-context learning" ability of LLMs.

Definition 5.3 (Induction heads)**.** *Let Σ be a finite alphabet and $w \in \Sigma^N$. For each $i \in [N]$, define*

$$\sigma(w, i) = \max\{\{0\} \cup \{j \in \mathbb{N} : j \leq i, w_{j-1} = w_i\}\}.$$

The induction head *task is to compute, for each $1 \leq i \leq N$, the value of $w_{\sigma(w,i)}$.*

For example, let $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and $w = \mathtt{aabcbabca}$. Then $w_{\sigma(w,9)} = \mathtt{b}$ because the 9th token is a, and the last occurrence of a before position 9 (which is in position 6) is followed by b.

The following theorem shows that our ANNA-transformer can solve induction heads problem using constant number of layers and sub-linear embedding dimension and number of heads.

Theorem 5.4. *Fix constants $0 < \varepsilon < \varepsilon' < 1$. There exists an ANNA-transformer T with $L = O(1)$ layers, $H = O(N^{(\varepsilon'-\varepsilon)/4})$ heads per layer, and embedding dimension $m = O(N^{\varepsilon'})$ such that $T(w)_i = w_{\sigma(w,i)}, \forall i \in [N]$, for all $w \in \Sigma^N$.*

We prove Theorem 5.4 (in Appendix F.2) by constructing a constant-round MPC algorithm for induction heads, and then applying Theorem 4.1 to convert it into an ANNA-transformer.

The induction heads task was generalized by [55] to a k-step variant called "k-hop".

Definition 5.5 (k-hop induction heads)**.** *Let Σ be a finite alphabet and $w \in \Sigma^N$. Let $\sigma^k(w, i)$ denote a k-fold composition of $\sigma(w, \cdot)$ from the previous definition. The* k-hop induction head *task is to compute $w_{\sigma^k(w,i)}$ for each $1 \leq i \leq N$.*

Using the same example where $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, $w = \mathtt{aabcbabca}$ and $k = 2$, we have $w_{\sigma(\sigma(w,9))} = \mathtt{a}$ because the last occurrence of b before position 7 is followed by a.

As was done in [55], we construct a $O(\log k)$-round MPC algorithm for k-hop using function composition, thus yielding a logarithmic depth scaling for ANNA-transformers on this task.

Theorem 5.6. *Fix constants $0 < \varepsilon < \varepsilon' < 1$, any $k \in \mathbb{N}$ and alphabet Σ with $|\Sigma| = O(N)$. There exists an ANNA-transformer T with $L = O(\log k)$ layers, $H = O(N^{(\varepsilon'-\varepsilon)/4})$ heads per layer, and embedding dimension $m = O(N^{\varepsilon'})$ such that $T(w)_i = w_{\sigma^k(w,i)}, \forall i \in [N]$, for all $w \in \Sigma^N$.*

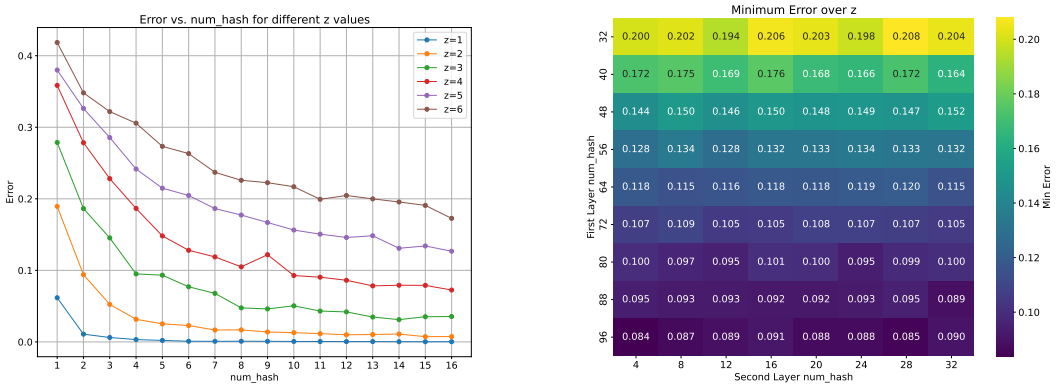
The full construction is given in Appendix F.2. We note that while prior works have given transformer constructions for k-hop [12, 55] (and Match2 [52]), these results do not directly imply ANNA-transformers constructions, given the differences in the architectures.

Prior work [55] showed that multi-layer recurrent nets and low-rank sub-quadratic attention [16, 32] are unable to solve k-hop unless the depth is $\Omega(k)$ or their memory size/embedding dimension is $\Omega(N/k^6)$. On contrast, ANNA-transformer achieves both $O(\log k)$-depth and sublinear (in N) width. In this sense, the k-hop task separates ANNA-transformer from these other efficient neural architectures.

5.3 Experiments on Match2 and induction heads

We empirically test the performance of ANNA-transformer on the Match2 and induction heads tasks. Experimental details are given in Appendix G. Since Algorithm 1 is not differentiable, we train a softmax version of attention as a surrogate, and then distill from the trained model to an ANNA-transformer (based on Algorithm 1 with angular LSH [6]). Our softmax attention normalizes all the queries and keys in $Q(X)$ and $K(X)$ to have unit norm, and computes $\mathrm{softmax}(\beta \cdot Q(X)K(X)^\intercal)V(X)$ with a tunable temperature parameter $\beta > 0$.

The Match2 dataset is generated the same way as [37] with context length $N = 32$ and upper bound $M = 37$. One-layer ANNA-transformers are able to achieve zero error with $\ell = 8$ hash tables and $z = 1$ hash function per table. See Figure 1a for the detailed performance. For induction heads, we use the dataset from [55] with number of hops $k = 1$, context length $N = 100$ and alphabet size $|\Sigma| = 4$. A two-layer ANNA-transformer achieved highly nontrivial error with $\ell = 32$ hash tables and $z = 2$ hash functions per table; with more hash tables, the error rate was as low as 0.1. See Figure 1b for the detailed results.



(a) Match2 (b) Induction heads

Figure 1: All errors are averaged over 10 runs. (a) Error rate on Match2: x-axis denotes the number of hash tables ℓ, and different colors correspond to different numbers z of hash functions per hash table. (b) Error rate on induction heads: Rows correspond to the number of hash tables in the first layer, columns correspond to the number of hash tables in the second layer. The reported error rate is the best achieved over the choice of $z \in \{1, 2, 3, 4\}$.

6 Conclusion and future work

In this work, we propose a more efficient class of neural architecture, the ANNA-transformers, which not only preserve the representational power of standard transformer characterized by the MPC framework but also yield a tighter equivalence with the MPC model. Furthermore, we show that constant layers of ANNA-transformers can simulate constant layers of low-rank transformer, and can solve reasoning tasks such as Match2 and k-hop tasks in near-optimal depth.

There are some interesting directions we leave for future work. While our ad hoc training method was effective as a proof-of-concept, it is desirable to develop a principled training method that directly optimizes the performance of an ANNA-transformer (or a differentiable variant thereof), rather than that of a surrogate model. Also, our empirical validation was limited to small synthetic datasets; extending these experiments to large-scale, real-world benchmarks is a promising next step.

7 Acknowledgements

Part of this work was done at the "Modern Paradigms in Generalization" and "Special Year on Large Language Models and Transformers, Part 1" programs at the Simons Institute for the Theory of Computing, Berkeley in 2024. We acknowledge support from the ONR under grants N00014-24-1-2700, N00014-22-1-2713, from the NSF under grant CCF2008733, and an award from the Columbia Center of AI Technology in collaboration with Amazon.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Josh Alman and Zhao Song. Fast attention requires bounded entries. *Advances in Neural Information Processing Systems*, 36:63117–63135, 2023.

[3] Josh Alman and Hantao Yu. Fundamental limitations on subquadratic alternatives to transformers. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=T2d0geb6y0`.

[4] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 459–468, 2006. doi: 10.1109/FOCS.2006.49.

[5] Alexandr Andoni, Aleksandar Nikolov, Krzysztof Onak, and Grigory Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 574–583, 2014.

[6] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 1225–1233, Cambridge, MA, USA, 2015. MIT Press.

[7] Dana Angluin. Local and global properties in networks of processors. In *STOC*, pages 82–93, 1980.

[8] Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, James Zou, Atri Rudra, and Christopher Re. Simple linear attention language models balance the recall-throughput tradeoff. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 1763–1840. PMLR, 21–27 Jul 2024.

[9] Paul Beame, Paraschos Koutris, and Dan Suciu. Communication steps for parallel query processing. *Journal of the ACM (JACM)*, 64(6):1–58, 2017.

[10] Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. On the Ability and Limitations of Transformers to Recognize Formal Languages. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7096–7116, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.576. URL https://aclanthology.org/2020.emnlp-main.576/.

[11] Satwik Bhattamishra, Michael Hahn, Phil Blunsom, and Varun Kanade. Separations in the representational capabilities of transformers and recurrent architectures. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[12] Alberto Bietti, Vivien Cabannes, Diane Bouchacourt, Herve Jegou, and Leon Bottou. Birth of a transformer: A memory viewpoint. *Advances in Neural Information Processing Systems*, 36: 1560–1588, 2023.

[13] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33*, 2020.

[14] Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. Scatterbrain: Unifying sparse and low-rank attention. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=SehIKudiIo1.

[15] Lijie Chen, Binghui Peng, and Hongxun Wu. Theoretical limitations of multi-layer transformer. *arXiv preprint arXiv:2412.02975*, 2024.

[16] 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*, 2021. URL https://openreview.net/forum?id=Ua6zuk0WRH.

[17] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492. URL https://doi.org/10.1145/1327452.1327492.

[18] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

[19] GeminiTeam, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[20] Mohsen Ghaffari. Massively parallel algorithms, June 2019. Lecture notes by Davin Choo, Computer Science, ETH Zurich.

[21] Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. Conditional hardness results for massively parallel computation from distributed lower bounds. *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1650–1663, 2019. URL `https://api.semanticscholar.org/CorpusID:201763923`.

[22] Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, 2023.

[23] Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. Sorting, searching, and simulation in the mapreduce framework. In *Proceedings of the 22nd International Conference on Algorithms and Computation*, ISAAC'11, page 374–383, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 9783642255908. doi: 10.1007/978-3-642-25591-5_39. URL `https://doi.org/10.1007/978-3-642-25591-5_39`.

[24] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[25] Michael Hahn. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171, December 2020. ISSN 2307-387X. doi: 10.1162/tacl_a_00306. URL `http://dx.doi.org/10.1162/tacl_a_00306`.

[26] MohammadTaghi Hajiaghayi, Silvio Lattanzi, Saeed Seddighin, and Cliff Stein. Mapreduce meets fine-grained complexity: Mapreduce algorithms for apsp, matrix multiplication, 3-sum, and beyond. *arXiv preprint arXiv:1905.01748*, 2019.

[27] Insu Han, Rajesh Jayaram, Amin Karbasi, Vahab Mirrokni, David Woodruff, and Amir Zandieh. Hyperattention: Long-context attention in near-linear time. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=Eh0Od2BJIM`.

[28] Yiding Hao, Dana Angluin, and Robert Frank. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Transactions of the Association for Computational Linguistics*, 10:800–810, 2022. ISSN 2307-387X. doi: 10.1162/tacl_a_00490. URL `http://dx.doi.org/10.1162/tacl_a_00490`.

[29] Sungjin Im, Ravi Kumar, Silvio Lattanzi, Benjamin Moseley, Sergei Vassilvitskii, et al. Massively parallel computation: Algorithms and applications. *Foundations and Trends® in Optimization*, 5(4):340–417, 2023.

[30] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, page 604–613, New York, NY, USA, 1998. Association for Computing Machinery. ISBN 0897919629. doi: 10.1145/276698.276876. URL `https://doi.org/10.1145/276698.276876`.

[31] Samy Jelassi, David Brandfonbrener, Sham M. Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 21502–21521. PMLR, 21–27 Jul 2024.

[32] Praneeth Kacham, Vahab Mirrokni, and Peilin Zhong. PolySketchFormer: Fast transformers via sketching polynomial kernels. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 22748–22770. PMLR, 21–27 Jul 2024.

[33] Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, page 938–948, USA, 2010. Society for Industrial and Applied Mathematics. ISBN 9780898716986.

[34] Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. On the computational complexity of self-attention. In *International Conference on Algorithmic Learning Theory*, 2023.

[35] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rkgNKkHtvB`.

[36] Alexander Kozachinskiy. Lower bounds on transformers with infinite precision. *arXiv preprint arXiv:2412.20195*, 2024.

[37] Alexander Kozachinskiy, Felipe Urrutia, Hector Jimenez, Tomasz Steifer, Germán Pizarro, Matías Fuentes, Francisco Meza, Cristian B. Calderon, and Cristóbal Rojas. Strassen attention: Unlocking compositional abilities in transformers based on a new lower bound method, 2025. URL `https://arxiv.org/abs/2501.19215`.

[38] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *STOC*, 1998.

[39] Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=3EWTEy9MTM`.

[40] Nathan Linial. Locality in distributed graph algorithms. *SIAM Journal on computing*, 21(1): 193–201, 1992.

[41] Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=De4FYqjFueZ`.

[42] Andreas Loukas. What graph neural networks cannot learn: depth vs width. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=B1l2bp4YwS`.

[43] William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023.

[44] William Merrill, Ashish Sabharwal, and Noah A Smith. Saturated transformers are constant-depth threshold circuits. *Transactions of the Association for Computational Linguistics*, 10: 843–856, 2022.

[45] William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In *Forty-first International Conference on Machine Learning*, 2024. URL `https://openreview.net/forum?id=QZgo9JZpLq`.

[46] John Miller and Moritz Hardt. Stable recurrent models. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=Hygxb2CqKm`.

[47] Moni Naor and Larry Stockmeyer. What can be computed locally? In *STOC*, pages 184–193, 1993.

[48] David Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

[49] Binghui Peng, Srini Narayanan, and Christos Papadimitriou. On limitations of the transformer architecture. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=KidynPuLNW.

[50] Tim Roughgarden, Sergei Vassilvitskii, and Joshua R Wang. Shuffles and circuits (on lower bounds for modern parallel computation). *Journal of the ACM*, 65(6):1–24, 2018.

[51] Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 02 2021. ISSN 2307-387X. doi: 10.1162/tacl_a_00353. URL https://doi.org/10.1162/tacl_a_00353.

[52] Clayton Sanford, Daniel Hsu, and Matus Telgarsky. Representational strengths and limitations of transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=36DxONZ9bA.

[53] Clayton Sanford, Bahare Fatemi, Ethan Hall, Anton Tsitsulin, Mehran Kazemi, Jonathan Halcrow, Bryan Perozzi, and Vahab Mirrokni. Understanding transformer reasoning capabilities via graph algorithms. *Advances in Neural Information Processing Systems*, 37:78320–78370, 2024.

[54] Clayton Sanford, Daniel Hsu, and Matus Telgarsky. One-layer transformers fail to solve the induction heads task. *arXiv preprint arXiv:2408.14332*, 2024.

[55] Clayton Sanford, Daniel Hsu, and Matus Telgarsky. Transformers, parallel computation, and logarithmic depth. In *Forty-First International Conference on Machine Learning*, 2024.

[56] Lena Strobl, William Merrill, Gail Weiss, David Chiang, and Dana Angluin. What formal languages can transformers express? a survey. *Transactions of the Association for Computational Linguistics*, 12:543–561, 2023. URL https://api.semanticscholar.org/CorpusID:264833196.

[57] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[58] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[59] Colin Wei, Yining Chen, and Tengyu Ma. Statistically meaningful approximation: a case study on approximating turing machines with transformers. In *Advances in Neural Information Processing Systems 35*, 2022.

[60] Kaiyue Wen, Xingyu Dang, and Kaifeng Lyu. RNNs are not transformers (yet): The key bottleneck on in-context retrieval. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=h3wbI8Uk1Z.

[61] Kai Yang, Jan Ackermann, Zhenyu He, Guhao Feng, Bohang Zhang, Yunzhen Feng, Qiwei Ye, Di He, and Liwei Wang. Do efficient transformers really save computation? In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 55928–55947. PMLR, 21–27 Jul 2024.

[62] Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. Self-attention networks can process bounded hierarchical languages. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics*, pages 3770–3785, 2021.

[63] Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. Are transformers universal approximators of sequence-to-sequence functions? In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=ByxRMONtvr`.

[64] Amir Zandieh, Insu Han, Majid Daliri, and Amin Karbasi. KDEformer: Accelerating transformers via kernel density estimation. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 40605–40623. PMLR, 23–29 Jul 2023.

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: We summarized all the results and contribution of the main paper in the our results section in the introduction.

   Guidelines:

   - The answer NA 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 NA 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: We have all the assumptions in the preliminaries part and briefly discuss the limitation about training in the last paragraph.

   Guidelines:

   - The answer NA 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: We have all the assumptions applicable to all the theorems in the preliminaries part and theorem-specific assumptions in the theorem statement. We have all the proofs in the Appendix.

Guidelines:

- The answer NA 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: [Yes]

   Justification: We provide all the experimental details in the Appendix G. Our main claims only focus on theoretical properties and experiments do not serve as the main contribution.

   Guidelines:

   - The answer NA 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: [No]

Justification: We will try to clean up the code and provide open access for the camera-ready version. Our main claims only focus on theoretical properties and experiments do not serve as the main contribution.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.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://nips.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, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide these design choices in the Appendix G.

   Guidelines:

   - The answer NA 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: [Yes]

   Justification: All the error reported are averaged over 10 runs and the variance is very small.

   Guidelines:

   - The answer NA 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: [Yes]

   Justification: We used 2 GPUs and provide the type of the GPU in Appendix G.

   Guidelines:

   - The answer NA 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: All the experiments are in the toy theoretical setting, and the data are all synthetic data which don't involve any human.

   Guidelines:

   - The answer NA 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: [NA]

    Justification: Our work focus understanding the theoretical properties of the computational model we proposed. We see the societal impart to be minimal.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA 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., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our experiments are on synthetic data, at most 2 layer small transformers, and do not have any pre-trained model.

    Guidelines:

    - The answer NA 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: We used the dataset from [55]. We cite this paper in the experiment section and follow the license.

    Guidelines:

    - The answer NA 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: [NA]

Justification: we do not release new assets.

Guidelines:

- The answer NA 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: [NA]

Justification: This is a theory paper and do not involve any human subject.

Guidelines:

- The answer NA 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: [NA]

Justification: This is a pure learning theory paper and do not involve these.

Guidelines:

- The answer NA 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 rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: LLMs are only used to assist writing and editing the paper.

Guidelines:

- The answer NA 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 (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Proof of Theorem 3.4

We restate Theorem 3.4 here, which gives the theoretical guarantee for Algorithm 1.

Theorem A.1 (LSH algorithm guarantee for ANNA; Theorem 3.4)**.** *Fix $c > \sqrt{3}$, LSH family $\mathcal{H}$ that is (r, cr, p_1, p_2)-sensitive with quality $\rho < 1/3$, $\ell = \Theta(N^{3\rho} \log N)$, and $z = \Theta(\log_{1/p_2} N)$. Then Algorithm 1 (with $\mathcal{H}$, ℓ, and z) implements an ANNA mechanism with parameters r, c, ℓ and $\eta = O(1/N^{1-3\rho})$.*[2]

Proof. Our algorithm applies for the regime with large approximation factor, i.e., $c > \sqrt{3}$. Since we only want the nearest neighbors within distance cr with the query point, we want to bound the probability of two points with distance greater than cr to fall into the same bucket. Consider the family G with $\Pr_{g\in G}[g(x) = g(y)] \leq \frac{0.1}{N^3}$, if $\|x - y\| > cr$. Then for each bucket, the expected number of collision (x, y fall into the same bucket and $\|x-y\| > cr$) is less than $N \cdot \Pr_{g\in G}[g(x) = g(y)] \leq \frac{0.1}{N^2}$. Therefore, by Markov's inequality, for each bucket, with probability greater than $1 - \frac{0.1}{N^2}$, there is no collision within the bucket. Then, by union bound over all the non-empty bucket (there are at most N of them), with probablity greater than $1 - \frac{0.1}{N}$, there is no collision in one hash table. By [30], $z = O(\log_{1/p_2} N)$, i.e., each hash function $g \in G$ is composed of $O(\log_{1/p_2} N)$ hash functions sampled from the LSH family $\mathcal{H}$, which suffices to achieve $\Pr_{g\in G}[g(x) = g(y)] \leq \frac{0.1}{N^3}$ whenever $\|x - y\| > cr$.

On the other hand, since the probability of collision is very small, the success probability (when $\|x-y\| \leq r$), namely $p = \Pr_{g\in G}[g(x) = g(y)] = N^{-3\rho}$ (recall that $\rho = \frac{\log 1/p_1}{\log 1/p_2}$), is also somewhat small. However, we can boost the success probability by using multiple hash tables. Let ℓ denote the number of hash tables. Then for each q_i, the probability of its r-nearest neighbor k ($k \in \mathcal{N}(q_i, r)$) falls into different bucket with q_i for all ℓ tables is upper bounded by $(1-p)^\ell$. By union bound over all possible nearest neighbors and all q_i's, the failure probability is bounded by $N^2(1-p)^\ell$. Assume we want the failure probability to be less than some $\delta > 0$, then we want $N^2(1-p)^\ell \leq \delta$. Taking logarithm of both sides, and using a Taylor expansion of $\log(1-x)$ for sufficiently small x, we find that $\ell = O(N^{3\rho}(\log N + \log 1/\delta))$ suffices for success probability $1 - \delta$.

Therefore, by union bound over all ℓ hash tables, with probability $1 - \frac{0.1}{N^{1-3\rho}}$, there is no collision in all the hash tables, which implies $w_{i,j} = 0$ if $\|k_j - q_i\| > cr$. By setting $\delta = \frac{0.1}{N^{1-3\rho}}$, we get $\ell = O(N^{3\rho} \log N)$. Hence, the total failure probability η is bounded by $\delta + \frac{0.1}{N^{1-3\rho}}$ which is $O(1/N^{1-3\rho})$.

If $\|k_j - q_i\| \leq r$, from the guarantee above, we know that k_j collides with q_i at least once in the ℓ hash bucket. This implies $w_{i,j} \geq 1/\text{count}$, where count is the number of all the collisions in the ℓ hash buckets that q_i retrieves. In the worst case, all the $k \in \mathcal{N}(q_i)$ collides with q_i in all ℓ hash tables except for k_j only colliding once. Therefore, count $\leq (\mathcal{N}(q_i) - 1) \cdot \ell$, and this gives us $w_{i,j} \geq \frac{1}{(\mathcal{N}(q_i)-1)\cdot\ell+1}$. □

Runtime and memory usage. One can see that for each query, we need to evaluate $O(N^{3\rho} \log_{1/p_2} N)$ hash functions and compute sum of m-dimensional vectors, so the total runtime is $O(mN^{1+3\rho} \log_{1/p_2} N)$. During the preprocessing, we need to store $N^{3\rho}$ hash tables and the sum of values, each with at most N buckets, so the total memory is $O(mN^{1+3\rho} \log N)$ bits. In fact, the space used can be further improved to $O(mN \log N)$ bits. Instead of maintaining ℓ hash tables, one can just store 1 hash table of size $O(mN \log N)$ with each entry responsible for tracking the values for each query. For each round of hashing (ℓ rounds in total), hash all queries using the hash functions and creates empty buckets for them. Then, hash each key, and if the key hashes to an existing query bucket, its value is added (along with a count). After processing keys, each query accumulates the values and counts from its corresponding bucket. We give the memory-efficient implementation in Algorithm 2.

[2] Optimal (data-oblivious) LSH schemes achieve $\rho = 1/c^2 + o(1)$ [4]. Since we assume $c > \sqrt{3}$, the failure probability $1/\text{poly}(N)$ decreases to zero with N.

Algorithm 2 Linear memory ANNA implementation with LSH family $\mathcal{H}$, ℓ hash tables, and z hash functions/table

Input: Input $X \in \mathbb{R}^{N \times d}$
Output: ANNA output for each of the query.

Let $q_i = Q(X)_i$, $k_i = K(X)_i$, and $v_i = V(X)_i$ for all $i \in [N]$.
Initialize an array of tuples A, and $A[i] \leftarrow (0, 0), \forall i \in [N]$.
for $u = 1$ to ℓ **do**
    Sample z hash functions $h_{u,1}, h_{u,2}, \ldots, h_{u,z}$ i.i.d. from $\mathcal{H}$.
    Create empty hash table T_u indexed by hash codes of queries (below).
    for each query q_i **do**
        Compute hash code $g_u(q_i) = (h_{u,1}(q_i), \ldots, h_{u,z}(q_i))$.
        Create an entry in T_u indexed by $g_u(q_i)$ and $T_u[g_u(q_i)] \leftarrow (0, 0)$.
    for each key-value pair (k_j, v_j) **do**
        Compute hash code $g_u(k_j) = (h_{u,1}(k_j), h_{u,2}(k_j), \ldots, h_{u,z}(k_j))$.
        if $T_u[g_u(k_j)]$ exists in T_u, **then** $T_u[g_u(k_j)]$ += $(v_j, 1)$.
    for each query q_i **do**
        $A[i]$ += $T_u[g_u(q_i)]$
Initialize a dictionary **attn** $\leftarrow \{(q_1, 0), (q_2, 0), \ldots, (q_N, 0)\}$.
for each query q_i **do**
    attn $\leftarrow A[i][0]/A[i][1]$
return attn

B ANNA-transformer can simulate MPC

Our simulation of MPC using ANNA-transformers uses only a special case of ANNA, which we call *Exact Match Attention (*EMA*)*. In EMA, we require the key to be *exactly the same as* the query for it to be considered in the attention matrix. We show that this special case already suffices to simulate MPC.

Definition B.1 (EM Attention)**.** *Let $X \in \mathbb{R}^{N \times d}$ be the input embedding, $Q, K, V : \mathbb{R}^{N \times d} \to \mathbb{R}^{N \times d}$ be query/key/value embedding functions. For any query q, let $\mathcal{N}(q) = \{k_j \in K : k_j = q\}$. For each query q_i, the* Exact Match attention *computes*

$$\mathrm{EMA}_{K,V}(q_i) = \begin{cases} \frac{1}{|\mathcal{N}(q_i)|} \sum_{j \in \mathcal{N}(q_i)} v_j & \text{if } \mathcal{N}(q_i) \neq \emptyset \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

EMA layer and EMA-transformer are defined analogously. To see that EMA is a special case of ANNA, notice that in ANNA, we can set $r = 0, c \to \infty$ and $w_{i,j} = \frac{1}{|\mathcal{N}(q_i)|}$ such that it becomes exactly the same as EM attention. EMA also admits a near linear-time algorithm: sort all the keys first (using a lexicographic ordering) in time $O(dN \log N)$ and space $O(dN)$; at query time, perform binary search in time $O(d \log N)$ per query.

We first give a simulation that directly simulates the R-round $(\varepsilon, \varepsilon)$-MPC using $L = R + 1$ layers but large embedding dimensions to showcase the core idea of the proof.

Theorem B.2 (EMA simulates MPC)**.** *For constant $0 < \varepsilon < 1$, any deterministic R-round MPC protocol π with N machines with $s = O(N^\varepsilon)$ words local memory, there exists an EMA-transformer T with depth $L = R+1$, number of heads $H = O(N^\varepsilon)$, and embedding dimension $m = O(N^{5\varepsilon} \log N)$, such that $T(\texttt{input}) = \pi(\texttt{input})$ for all $\texttt{input} \in \mathbb{Z}_{2^p}^N$.*

Proof. For any R-round MPC protocol π with N machines that maps the `input` to `output`, we define the intermediate steps for local computation phase and message transimission phase. We denote the input to all the machines before the local computation as $\texttt{MachineIn}_1, \texttt{MachineIn}_2, \ldots, \texttt{MachineIn}_\texttt{R}$, and denote the information after deterministic local computations $(\texttt{Local}_\texttt{r}^\texttt{i})_{\texttt{r} \in [\texttt{R}], \texttt{i} \in [\texttt{N}]}$ as $\texttt{MachineOut}_1, \texttt{MachineOut}_2, \ldots, \texttt{MachineOut}_\texttt{R}$, where $\texttt{MachineOut}_\texttt{r}^\texttt{i} = \texttt{Local}_\texttt{r}^\texttt{i}(\texttt{MachineIn}_\texttt{r}^\texttt{i})$. In the communication (message transimission) phase, we need to route the messages to the correct machines ie from $\texttt{MachineOut}_\texttt{r}$ to $\texttt{MachineIn}_{\texttt{r}+1}$.

In our simulation, each token input to the EMA-transformer plays the role of a machine in the MPC protocol. We simulate the local computation functions $(\texttt{Local}_{\texttt{r}}^{\texttt{i}})_{\texttt{r}\in[\texttt{R}],\texttt{i}\in[\texttt{N}]}$ by the element-wise functions $Q(\cdot), K(\cdot), V(\cdot)$ in the architecture. Therefore, the simulation process can be partitioned into 3 different parts:

1. Initialization. The `input` feeded into EMA-transformer is distributed in the N tokens, and we need to transfer than into the first $\lceil \frac{N}{s} \rceil$ tokens/machines to match $\texttt{MachineIn}_1$.
2. Routing (message transmission). After the local computation in each round r, we need to communicate the messages from $\texttt{MachineOut}_{\texttt{r}}$ to $\texttt{MachineIn}_{\texttt{r}+1}$.
3. Final output. The MPC `output` is distributed in the first $\lceil \frac{N}{s} \rceil$ tokens/machines, and we need to distributed them back to the N tokens.

The following 3 lemmas construct the elements for each of these 3 parts.

We first show the message transmission part of MPC can be simulate by the EMA-transformer. Recall that after r rounds of local computation, each machine i has a set of messages it wants to send to other machines, denoted by $\texttt{MachineOut}_{\texttt{r}}^{\texttt{i}} = \{(\texttt{Msg}_{\texttt{dest}}^{\texttt{i}}, \texttt{dest}) : \texttt{dest} \in \texttt{sent}^{\texttt{i}}\}$, where $\texttt{sent}^{\texttt{i}}$ is the set of machine indices that machine i will sent the message to and $\texttt{Msg}_{\texttt{dest}}^{\texttt{i}}$ is the message machine i send to machine `dest`. After the message communication phase, each machine i has the set of messages it receives from other machines, denoted by $\texttt{MachineIn}_{\texttt{r}+1}^{\texttt{i}} = \{(\texttt{Msg}, \texttt{Src}) : (\texttt{Msg}, \texttt{i}) \in \texttt{MachineOut}_{\texttt{r}}^{\texttt{Src}}\}$. Since each machine can only send/receive s words, we have $\sum_{\texttt{dest}\in\texttt{sent}^{\texttt{i}}} |\texttt{Msg}| \leq \texttt{s}$ and $\sum_{(\texttt{Msg},\texttt{i})\in\texttt{MachineOut}_{\texttt{r}}^{\texttt{Src}}} |\texttt{Msg}| \leq \texttt{s}$ for all machine i. We call this process the routing process of MPC. The following lemma shows that each routing round of MPC can be simulated by one layer of EMA-transformer.

Lemma B.3 (Routing)**.** *For any R-round MPC protocol π having q machines each with local memory s and any $r \in [R-1]$, there exists an EMA-transformer $\texttt{route}_{\texttt{r}}$ with $H = O(s)$ heads and $m = O(\log q)$ for Q and K, $m = O(s^5 \log q)$ for V that takes input $X = \texttt{MachineIn}_{\texttt{r}}$ and produces output $\texttt{route}_{\texttt{r}}(\texttt{X}) = \texttt{MachineIn}_{\texttt{r}+1}$.*

Proof. Follow the assumption in [55], we encode the local computation into the element-wise operations $Q(\cdot), K(\cdot), V(\cdot)$ of transformer. The main part of the proof will focus on using EMA to route $\texttt{MachineOut}_{\texttt{r}}$ to $\texttt{MachineIn}_{\texttt{r}+1}$.

We assign a unique positional encoding or identifier to each machine i, denoted by p_i. This can be done with $O(\log q)$ bits. This encoding serves as a unique key to retrieve the message in each machine. The high level idea is to create a query for each machine i and a key for each $\texttt{dest} \in \texttt{sent}_{\texttt{i}}$ and the associated value is the message $\texttt{Msg}_{\texttt{dest}}^{\texttt{i}}$ sent to `dest` in the protocol. Since each machine can send at most s messages to other machines, we create s EMA heads and each head is responsible for one message for all the q machines. Each machine retrieves the message sent to them by having a query in each head. Because each query can attend only to perfectly matching keys, each distinct outbound message must be passed by a different attention head, but multiple inbound messages may be received by the same attention head.

Specifically, let Q^h, K^h, V^h be the query, key, value embedding after the machine local computation for each head $h \in [s]$. Set $q_i^h = p_i$ for all h, so

$$Q^1 = Q^2 = \cdots = Q^s = \begin{pmatrix} p_1^{\mathsf{T}} \\ p_2^{\mathsf{T}} \\ \vdots \\ p_q^{\mathsf{T}} \end{pmatrix}.$$

Let $k_i^h = p_{\texttt{dest}_{\texttt{h}}^{\texttt{i}}}$ for $\texttt{dest}_{\texttt{h}}^{\texttt{i}} \in \texttt{sent}^{\texttt{i}} = \{\texttt{dest}_{\texttt{1}}^{\texttt{i}}, \texttt{dest}_{\texttt{2}}^{\texttt{i}}, \ldots, \texttt{dest}_{\texttt{s}}^{\texttt{i}}\}$, where $\texttt{dest}_{\texttt{j}}^{\texttt{i}}$ is the destination machine index for the jth word message that machine i sends. The key matrices are constructed as follows:

$$K^1 = \begin{pmatrix} p_{\texttt{dest}_1^1}^{\mathsf{T}} \\ p_{\texttt{dest}_1^2}^{\mathsf{T}} \\ \vdots \\ p_{\texttt{dest}_1^{\texttt{q}}}^{\mathsf{T}} \end{pmatrix}, \quad K^2 = \begin{pmatrix} p_{\texttt{dest}_2^1}^{\mathsf{T}} \\ p_{\texttt{dest}_2^2}^{\mathsf{T}} \\ \cdots \\ p_{\texttt{dest}_2^{\texttt{q}}}^{\mathsf{T}} \end{pmatrix}, \quad \ldots, \quad K^s = \begin{pmatrix} p_{\texttt{dest}_s^1}^{\mathsf{T}} \\ p_{\texttt{dest}_s^2}^{\mathsf{T}} \\ \vdots \\ p_{\texttt{dest}_s^{\texttt{q}}}^{\mathsf{T}} \end{pmatrix}.$$

Let v_i^h be some embedding of $(\mathtt{Msg}^{\mathtt{i}}_{\mathtt{dest}^{\mathtt{i}}_{\mathtt{h}}}, \mathtt{dest}^{\mathtt{i}}_{\mathtt{h}}, \mathtt{i})$, denoted by $v_i^h = \mathtt{emb}^{\mathtt{h}}_{\mathtt{i}}(\mathtt{Msg}^{\mathtt{i}}_{\mathtt{dest}^{\mathtt{i}}_{\mathtt{h}}}, \mathtt{dest}^{\mathtt{i}}_{\mathtt{h}}, \mathtt{i})$ for some $\mathtt{emb}^{\mathtt{h}}_{\mathtt{i}}$ defined later, and

$$V^1 = \begin{pmatrix} \mathtt{emb}^1_1(\mathtt{Msg}^1_{\mathtt{dest}^1_1}, \mathtt{dest}^1_1, 1) \\ \mathtt{emb}^1_2(\mathtt{Msg}^2_{\mathtt{dest}^2_1}, \mathtt{dest}^2_1, 2) \\ \vdots \\ \mathtt{emb}^1_{\mathtt{q}}(\mathtt{Msg}^{\mathtt{q}}_{\mathtt{dest}^{\mathtt{q}}_1}, \mathtt{dest}^{\mathtt{q}}_1, \mathtt{q}) \end{pmatrix}, \quad \ldots, \quad V^s = \begin{pmatrix} \mathtt{emb}^{\mathtt{s}}_1(\mathtt{Msg}^1_{\mathtt{dest}^1_{\mathtt{s}}}, \mathtt{dest}^1_{\mathtt{s}}, 1) \\ \mathtt{emb}^{\mathtt{s}}_2(\mathtt{Msg}^2_{\mathtt{dest}^2_{\mathtt{s}}}, \mathtt{dest}^2_{\mathtt{s}}, 2) \\ \vdots \\ \mathtt{emb}^{\mathtt{s}}_{\mathtt{q}}(\mathtt{Msg}^{\mathtt{q}}_{\mathtt{dest}^{\mathtt{q}}_{\mathtt{s}}}, \mathtt{dest}^{\mathtt{q}}_{\mathtt{s}}, \mathtt{q}) \end{pmatrix}.$$

By such construction of Q, K, V, in our EMA, each query will retrieve the average value of the messages whose key exactly matches the query. However, by setting the value matrix this way, we might corrupt the message when there are more than one $k_i^h \in K^h$ that are equal to the same query. To solve this problem, we can apply the same *multiple hashing-based encoding* in Lemma 3.2 from [55], which encodes each message in multiple fixed locations generated by a sparse binary matrix and have an extra "validity bit" indicating whether the message is corrupted or not. We restate an adapted version of their Lemma 3.2 here.

Lemma B.4 (Lemma 3.2 of [55]; message encoding in sparse averaging)**.** *For any message size $\Delta \in \mathbb{N}$, message count bound $\alpha \in \mathbb{N}$, there exist an encoding function ϕ such that ϕ takes in $(\mathtt{Msg}^{\mathtt{i}}_{\mathtt{dest}^{\mathtt{i}}_{\mathtt{h}}}, \mathtt{dest}^{\mathtt{i}}_{\mathtt{h}}, \mathtt{i})$ defined above where the size of it is bounded by Δ for all $i \in [q]$ and $h \in [\alpha]$ and encodes it into $\mathtt{emb}^{\mathtt{h}}_{\mathtt{i}}(\mathtt{Msg}^{\mathtt{i}}_{\mathtt{dest}^{\mathtt{i}}_{\mathtt{h}}}, \mathtt{dest}^{\mathtt{i}}_{\mathtt{h}}, \mathtt{i}) \in \mathbb{R}^{\mathtt{m}}$ with $m = O(\alpha^4 \Delta \log q)$, and a decoder function φ such that φ takes in the output of the EMA with Q, K, V defined above and decodes it into $(\mathtt{Msg}^{\mathtt{i}}_{\mathtt{dest}^{\mathtt{i}}_{\mathtt{h}}}, \mathtt{dest}^{\mathtt{i}}_{\mathtt{h}}, \mathtt{i})$.*

Let $\mathtt{rcvd}^{\mathtt{i}} = \{\mathtt{src}^{\mathtt{i}}_1, \mathtt{src}^{\mathtt{i}}_2, \ldots, \mathtt{src}^{\mathtt{i}}_{\mathtt{s}}\}$, where $\mathtt{src}^{\mathtt{i}}_{\mathtt{j}}$ is the jth source machine index that machine i receives message from. Because $|\mathtt{sent}^{\mathtt{i}}| \le \mathtt{s}$ and $|\mathtt{rcvd}^{\mathtt{i}}| \le \mathtt{s}$, in each head of EMA, there are at most s values get retrieved and averaged for each query. Thus, here we can just apply Lemma B.4 with $\alpha = \Delta = s$, which gives us an embedding dimension bound $m = O(s^5 \log q)$. □

We then show with one layer EMA-transformer, we can properly initialize the setup of MPC by converting $\text{Input} = (\mathtt{input}_1, \mathtt{input}_2, \ldots, \mathtt{input}_{\mathtt{n}})$ to $\mathtt{MachineIn}_1$, the input before the first round of MPC computation, ie the input is distributed evenly on the first $\lceil \frac{n}{s} \rceil$ machines.

Lemma B.5 (Initialization)**.** *For any R-round MPC protocol π having q machines each with local memory s and n-word input, there exists an EMA-transformer* $\mathtt{init}$ *with $H = 1$ head, $m = O(\log q)$ for Q, K and $m = O(s)$ for V that takes* $\mathtt{input}$ *and outputs* $\mathtt{init}(\mathtt{input}) = \mathtt{MachineIn}_{\mathtt{r}}$.

Proof. The input should be distributed accross each machine $1 \le i \le \lceil \frac{n}{s} \rceil$ with $\mathtt{MachineIn}^{\mathtt{i}}_1 = \{(\mathtt{input}_{\mathtt{idx}}, \mathtt{idx}) : \mathtt{idx} \in \mathtt{s(i-1)+1}, \ldots, \min\{\mathtt{n}, \mathtt{si}\}\}$. Let $q_{in} = \lceil \frac{n}{s} \rceil$ be the number of machines to store the initial input. Since the input given to $\mathtt{init}$ are n tokens (here we treat each token as a machine), we need to rearange the memory so that the input is distributed on the first q_{in} tokens.

Same as before, we use the positional encoding p_i to be the unique identifier for each machine. We create a key value pair for each input token and the key corresponds to the identifier of the machine that $\mathtt{input}_{\mathtt{idx}}$ goes to and the value be $(\mathtt{input}_{\mathtt{idx}}, \mathtt{idx})$. Also, create a query for each machine $i \in [q_{in}]$.

For each machine $i \in [q_{in}]$, define the query embedding $q_i = p_i$,

$$Q = \begin{pmatrix} p_1^\intercal \\ p_2^\intercal \\ \vdots \\ p_{q_{in}}^\intercal \end{pmatrix}$$

For each token $\mathtt{input}_{\mathtt{idx}}, \mathtt{idx} \in [\mathtt{n}]$, let $\mathtt{dest}_{\mathtt{idx}} = \lceil \frac{\mathtt{idx}}{\mathtt{s}} \rceil$ be the machine storing the token, define the key embedding $k_{\mathtt{idx}} = p_{\mathtt{dest}_{\mathtt{idx}}}$,

$$K = \begin{pmatrix} p^\intercal_{\mathtt{dest}_1} \\ p^\intercal_{\mathtt{dest}_2} \\ \vdots \\ p^\intercal_{\mathtt{dest}_{\mathtt{n}}} \end{pmatrix}$$

Let $i' = \texttt{idx} \mod \texttt{s}$. For each token $\texttt{input}_{\texttt{idx}}, \texttt{idx} \in [\texttt{n}]$, define the value embedding $v_{\texttt{idx}} \in \mathbb{R}^{2s}$ to be $(\texttt{input}_{\texttt{idx}}, \texttt{idx})$ in the $2i'-1, 2i'$-th entry and 0 in all other entry,

$$V = \begin{pmatrix} \texttt{input}_1 & 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & \texttt{input}_2 & 2 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & \texttt{input}_{\texttt{s}} & s \\ \texttt{input}_{\texttt{s}+1} & s+1 & 0 & 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix}$$

By setting the value matrix like this, we can avoid corrupting the messages. □

Last, we show that with an additional one layer EMA-transformer, we can map the final round $\texttt{MachineIn}_{\texttt{R}}$ to the $\texttt{output}$ of MPC protocol where the output is store in the first $\lceil \frac{n}{s} \rceil$ machines.

Lemma B.6 (Final Output)**.** *For any R-round MPC protocol π having q machines each with local memory s and n-word input, there exists an EMA-transformer* $\texttt{out}$ *with $H = 1$ head, $m = O(\log q)$ for Q, K and $m = O(s)$ for V that takes* $\texttt{MachineIn}_{\texttt{R}}$ *and outputs* $\texttt{out}(\texttt{MachineIn}_{\texttt{R}})_{:\texttt{n}} = \pi(\texttt{input}) = \texttt{output}$.

Proof. First, the element-wise operations can compute $\texttt{MachineOut}_{\texttt{R}}$ from $\texttt{MachineIn}_{\texttt{R}}$. The output is distributed accross each machine $1 \le i \le \lceil \frac{n}{s} \rceil = q_{out}$ with memory of machine i be $\texttt{output}^{\texttt{i}} = \{(\texttt{output}_{\texttt{idx}}, \texttt{idx}) : \texttt{idx} \in \texttt{s}(\texttt{i}-1)+1, \dots, \min\{\texttt{n}, \texttt{si}\}\}$. Then, we just need to retrieve the output tokens from all the q_{out} machines and distribute them back to n tokens. This step does the inverse job of init. We create a query for each token $\texttt{output}_{\texttt{idx}}$ for all $\texttt{idx} \in [\texttt{n}]$. Let $\texttt{src}_{\texttt{idx}} = \lceil \frac{\texttt{idx}}{\texttt{s}} \rceil$ be the machine token $\texttt{output}_{\texttt{idx}}$ is in.

For each token $\texttt{output}_{\texttt{idx}}, \texttt{idx} \in [\texttt{n}]$, define the query embedding $q_{\texttt{idx}} = p_{\texttt{src}_{\texttt{idx}}}$,

$$Q = \begin{pmatrix} p_{\texttt{src}_1}^\intercal \\ p_{\texttt{src}_2}^\intercal \\ \vdots \\ p_{\texttt{src}_{\texttt{n}}}^\intercal \end{pmatrix}$$

For each machine $i \in [q_{out}]$, create a key $k_i = p_i$,

$$K = \begin{pmatrix} p_1^\intercal \\ p_2^\intercal \\ \vdots \\ p_{q_{out}}^\intercal \end{pmatrix}$$

The value associates with each key i is the memory $\texttt{MsgOut}_{\texttt{i}}$ stored in each machine i. Define the value embedding $v_i = \texttt{MsgOut}_{\texttt{i}}$,

$$V = \begin{pmatrix} \texttt{MsgOut}_1 \\ \texttt{MsgOut}_2 \\ \vdots \\ \texttt{MsgOut}_{\texttt{q}_{\texttt{out}}} \end{pmatrix}$$

By choosing a proper element-wise function φ, $\text{out}(\texttt{MachineIn}_{\texttt{R}})_{\texttt{i},1} = \texttt{output}_{\texttt{i}}$. □

The theorem follows from stacking the elements from these three lemmas. Each lemma gives us a single layer of the final EMA-transformer T with embedding dimension $m = O(N^{5\varepsilon} \log N)$:

$$T = \texttt{out} \circ \texttt{route}_{\texttt{R}-1} \circ \cdots \circ \texttt{route}_1 \circ \texttt{init}$$ □

Remark B.7 (General (γ, ε)-MPC)**.** *The above simulation works the same for (γ, ε)-MPC by padding $\max(0, O(N^{1+\gamma-\varepsilon}) - N)$ empty chain-of-thought tokens in the input.*

Remark B.8 (Number of heads)**.** *The standard transformer can simulate MPC using the same embedding dimension but only 1 attention head [53, 55]. Here the EMA needs $O(N^\varepsilon)$ heads, and we leave how to improve the number of heads for future work.*

Since EM attention is a special case of ANN attention with $r = 0$ and $c \to \infty$, the simulation of MPC with ANNA-transformer naturally follows from Theorem B.2.

Corollary B.9 (ANNA simulates MPC)**.** *For constant $0 < \varepsilon < 1$, any deterministic R-round MPC protocol π with N machines with $s = O(N^\varepsilon)$ words local memory, there exists an ANNA-transformer T with depth $L = R + 1$, number of heads $H = O(N^\varepsilon)$, and embedding dimension $m = O(N^{5\varepsilon} \log N)$, such that $T(\texttt{input}) = \pi(\texttt{input})$ for all $\texttt{input} \in \mathbb{Z}_{2^p}^N$.*

Theorem B.2 only gives us an MPC simulation in the sublinear local memory regime when $s = O(N^{1/5-\delta})$ for any $\delta > 0$. However, a lot of MPC protocol algorithms require $s = \Omega(N^{1/2})$ local, such as MPC algorithm for 3-SUM [26] and algorithms for graphs [53]. The above simulation using EMA-transformer does not yield a sublinear embedding dimension. [53] further gives a simulation of MPC with sublinear local memory using transformer with sublinear embedding dimension by simulating one round of MPC protocol with $O(1)$ layers of transformer, instead of just one layer. Their improvement also applies here.

Theorem B.10 (EMA simulates MPC with improved embedding dimension)**.** *For constant $0 < \varepsilon < \varepsilon' < 1$, any deterministic R-round MPC protocol π with N machines with $s = O(N^\varepsilon)$ words local memory, there exists an EMA-transformer of depth $L = O(R)$, number of heads $H = O(N^{\frac{\varepsilon'-\varepsilon}{4}})$ and embedding dimension $m = O(N^{\varepsilon'})$ such that $T(\texttt{input}) = \pi(\texttt{input})$ for all $\texttt{input} \in \mathbb{Z}_{2^p}^N$.*

Proof. The proof relies on simulating any MPC protocol by a restricted version of MPC protocol [53] which limits the number of machines each machine can send message to. Then, use a modified version of Theorem B.2 to simulate this restricted version of MPC.

Definition B.11 (Definition 3 of [53])**.** *For constants $\gamma, \varepsilon, \rho > 0$, a $(\gamma, \varepsilon, \rho)$-MPC protocol is a (γ, ε)-MPC protocol with an additional constraint: in each round, each machine can only send/receive messages from $k = O(n^\rho)$ machines, while the total size of messages it can send and receive is still $s = O(N^\varepsilon)$. We refer to k as the* communication capacity*.*

[53] gives a construction that simulates a R-round (γ, ε)-MPC protocol with $O(R)$-round $(\gamma, \varepsilon, \rho)$-MPC protocol. We restate their proposition here.

Lemma B.12 (Proposition 24 of [53]; $(\gamma, \varepsilon, \rho)$-MPC simulates (γ, ε)-MPC)**.** *For constants $\gamma, \varepsilon > 0$ and $\rho \in (0, \varepsilon/2)$, if function f can be computed by an R-round (γ, ε)-MPC, it can also be computed by a $O(\frac{R(1+\gamma)^2}{\rho^2})$-round $(\gamma, \varepsilon, \rho)$-MPC protocol.*

Therefore, we just need to simulate $(\gamma, \varepsilon, \rho)$-MPC protocol using our EMA-transformer. The simulation follows the same recipe as Thereom B.2, where we have the initialization, message passing and final output phase. Since the initialization and output of $(\gamma, \varepsilon, \rho)$-MPC follow the same rule as (γ, ε)-MPC, we only need to modify the message passing part of the simulation which corresponds to the routing Lemma B.3.

Lemma B.13 (EMA simulates $(\gamma, \varepsilon, \rho)$-MPC)**.** *For constant $0 < \rho < \varepsilon < 1$, any deterministic R-round MPC protocol π with N machines with $s = O(N^\varepsilon)$ words local memory and communication capacity $k = O(N^\rho)$, there exists an EMA-transformer of depth $L = R + 1$, number of heads $H = O(N^\rho)$ and embedding dimension $m = O(N^{\varepsilon+4\rho} \log N)$ such that $T(\texttt{input}) = \pi(\texttt{input})$ for all $\texttt{input} \in \mathbb{Z}_{2^p}^N$.*

Proof. For the initialization and final output part, we just use the same $\texttt{init}$ and $\texttt{out}$ constructed in Lemma B.5 and Lemma B.6. In the routing part (Lemma B.3), because $|\texttt{sent}^\texttt{i}| \le \texttt{k}, |\texttt{rcvd}^\texttt{i}| \le \texttt{k}$, we only need k heads and in each head of EMA, there are at most k keys matching each query and thus at most k values get averaged for a single query. Therefore, we can apply Lemma B.4 with $\alpha = k$ and $\Delta = s$, leading to an embedding dimension $m = O(N^{\varepsilon+4\rho} \log N)$. This gives us a new $\texttt{route}'_\texttt{r}$ with reduced number of heads and embedding dimension for each round r.

Likewise, we stack the 3 building blocks of one-layer EMA-transformer and have an $(R+1)$-layer EMA transformer

$$T = \texttt{out} \circ \texttt{route}'_{\texttt{R}-1} \circ \cdots \circ \texttt{route}'_1 \circ \texttt{init}$$

and this finishes the construction for the lemma. □

Let $\rho = \min(\varepsilon/2, (\varepsilon' - \varepsilon)/4)$. In this setting, $\gamma = \varepsilon$. By Lemma B.12, we can simulate the R-round (γ, ε)-MPC by an $R' = O(\frac{R(1+\varepsilon)^2}{\min(\varepsilon^2, (\varepsilon'-\varepsilon)^2)})$-round $(\gamma, \varepsilon, \rho)$-MPC. Then, by Lemma B.13, we can simulate this R'-round $(\gamma, \varepsilon, \rho)$-MPC by an $R' + 1$-layer EMA transformer with $O(N^\rho)$ heads and embedding dimension $O(N^{\varepsilon+4\rho} \log N) = O(N^{\varepsilon'})$. □

Again, the improved simulation result of ANNA-transformer follows from Theorem B.10.

Corollary B.14 (ANNA simulates MPC with improved embedding dimension)**.** *For constant $0 < \varepsilon < \varepsilon' < 1$, any deterministic R-round MPC protocol π with N machines with $s = O(N^\varepsilon)$ words local memory, there exists an ANNA-transformer of depth $L = O(R)$, number of heads $H = O(N^{\frac{\varepsilon'-\varepsilon}{4}})$ and embedding dimension $m = O(N^{\varepsilon'})$ such that $T(\texttt{input}) = \pi(\texttt{input})$ for all $\texttt{input} \in \mathbb{Z}_{2^p}^N$.*

C MPC can Simulate ANNA-transformer

As a warm-up, we first simulate EMA-transformers using MPC, and then generalize it to the simulation of ANNA-transformers.

Theorem C.1 (MPC simulates EMA)**.** *Fix constants $0 < \varepsilon < \varepsilon' < 1$. For any L-layer EMA-transformer T with $mH = O(N^\varepsilon)$, there exists a $O(\frac{L}{\varepsilon'-\varepsilon})$-round MPC protocol π with local memory $s = O(N^{\varepsilon'})$ and $P = O(N^{1+\varepsilon-\varepsilon'})$ machines such that $\pi(\texttt{input}) = T(\texttt{input})$ for all $\texttt{input} \in \mathbb{Z}_{2^p}^N$.*

Proof. We first show how to use MPC to simulate one layer of EMA-transformer. In the high level, for each token x_i, we have a token Machine i which is responsible for computing the key, query and value embedding for x_i and other element-wise computation on x_i. The main bulk of the proof is to search for the exact matching keys for each query and send the averaged values associated with the matching keys to the token machines. In order to do this, we sort all the key and value pairs (k_i, v_i) in the order defined by the key. We divide the sorted key and value pairs into buckets such that each bucket contains the same keys. For each bucket, we have a "meta-info" machine to store the indices of the machines that contains the keys in the bucket. We then compute the averaged values within each bucket and store the averaged value into the "meta-info" machine and propagate the value to all the queries that match with the key.

To begin with, let $\mathcal{X}$ denote the space of query and key, and we define a comparator $<$ over $\mathcal{X}$ in order to sort. Without loss of generality, we just define it to be the lexicographical ordering comparator. Based on this comparator, we define a query ranking permutation of $[N]$ by $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_N)$ and a key ranking permutation of $[N]$ by $\sigma' = (\sigma'_1, \sigma'_2, \ldots, \sigma'_N)$ such that

$$\begin{aligned} q_{\sigma_1} < q_{\sigma_2} < \cdots < q_{\sigma_N} \\ \text{and } k_{\sigma'_1} < k_{\sigma'_2} < \cdots < k_{\sigma'_N} \end{aligned}$$

For the"meta-info" machine, we use a uniform hash function $h : \mathcal{X} \to [N]$ to map queries and keys to their corresponding "meta-info" machine. Recall that for a uniform hash function h, $\mathbb{P}(h(a) = h(b)) = \frac{1}{N}$, for any $a, b \in \mathcal{X}$ and $a \neq b$. Therefore,

$$\begin{aligned} &\mathbb{P}(\exists i \text{ such that the size of bucket } h(q_i) \geq s) \\ &\leq \mathbb{P}(\exists s \text{ different elements fall into one bucket}) \\ &\leq \binom{N}{s} \frac{1}{N^s} \leq \frac{1}{s!} = \frac{1}{N^{\varepsilon'}!} \end{aligned}$$

With high probability, each "meta-info" machine is responsible for at most s keys or queries.

We divide the machines into different types and summarize the role of each type of machine here:

- For $i \in [N]$, Machine i is the *token machine* for x_i. This machine performs all the element-wise computation for token i. Specifically, it computes the query, key, value embeddings q_i, k_i, v_i and element-wise operations after the attention layer.

- For $i \in [\lceil mN/s \rceil]$, Machine (i, Q) is a data structure machine for sorting queries and storing the ith chunk of the sorted list of queries after sorting. In other words, let $n_q = \lfloor s/m \rfloor$ be the number of queries each machine can store and, at the end of sorting, machine (i, Q) stores $\{q_{\sigma_{(i-1)\cdot n_q+1}}, \ldots, q_{\sigma_{i\cdot n_q}}\}$.
- For $i \in \lceil 2mN/s \rceil$, Machine (i, KV) is a data structure machine for sorted list of key and value pairs. In other words, let $n_k = \lfloor s/2m \rfloor$ be the number of key and value pairs each machine can store and, at the end of sorting, machine (i, KV) stores $\{(k_{\sigma'_{(i-1)\cdot n_q+1}}, v_{\sigma'_{(i-1)\cdot n_q+1}}), \ldots, (k_{\sigma'_{i\cdot n_q}}, v_{\sigma'_{i\cdot n_q}})\}$.
- For $i \in [N]$, Machine (i, h_q) is the "meta-info" machine for the queries whose hash value is i. Let $h_q^i = \{q_j | j \in [N], h(q_j) = i\}$. This machine stores the location information of $q \in h_q^i$ in the sorted list. Specifically, for all $q \in h_q^i$, this machine stores the start machine index, i.e. (start, Q) where start $= \arg\min_j \{q \in$ Machine $(j, Q)\}$, and the end machine index, i.e. (end, Q) where end $= \arg\max_j \{q \in$ Machine $(j, Q)\}$.
- For $i \in [N]$, Machine (i, h_k) is the "meta-info" machine for the keys whose hash value is i. Let $h_k^i = \{k_j | j \in [N], h(k_j) = i\}$. This machine stores the location information of $k \in h_k^i$ in the sorted list. Specifically, for all $k \in h_q^i$, this machine stores the start machine index, i.e. (start, KV) where start $= \arg\min_j \{k \in$ Machine $(j, KV)\}$, and the end machine index, i.e. (end, Q) where end $= \arg\max_j \{k \in$ Machine $(j, KV)\}$.
- The auxiliary machines needed for message propagation.

We proceed to discuss the MPC protocol for computing the output of one layer single head EM-attention transformer. In the first round, (same as the token dispersion stage of [55]), route each token x_i to its corresponding token machine i.

In the second round, each token machine i computes the query, key value embedding $q_i = Q(x_i), k_i = K(x_i), v_i = V(x_i)$ and sends (q_i, i) to the sorting query data structure machine $(\lceil mi/s \rceil, Q)$ and (k_i, v_i, i) to the sorting key data structure machine $(\lceil 2mi/s \rceil, KV)$.

Then, sorting query data structure machines ((i, Q) for all $i \in \lceil mN/s \rceil$) sorts the queries. Sorting in MPC has been well studied, and this can be done in constant number of rounds [23].

Lemma C.2 (MPC Sorting)**.** *There exists an MPC protocol with local memory $s = O(N^{\varepsilon'})$ that can sort N items and each item has size $O(N^{\varepsilon}), \varepsilon < \varepsilon'$ in $O(\frac{1}{\varepsilon'-\varepsilon})$ rounds with $O(N^{1+\varepsilon-\varepsilon'})$ machines.*

After sorting, for each $i \in [N]$, we need to send the location information of q_i, ie which data structure machines contains q_i, to its "meta-info" machine $(h(q_i), h_q)$. The idea is to build an $\frac{s}{m}$-ary tree structure to aggregate the information and each query data structure machine is a leaf node of this tree. Recall that each machine (i, Q) stores the queries $S = \{q_{\sigma_{(i-1)\cdot n_q+1}}, \ldots, q_{\sigma_{i\cdot n_q}}\}$. If S contains the start and end of a particular query vector q_l, then (i, Q) sends a message $(q_l, (i, Q))$ to machine $(h(q_l), h_q)$. Machine (i, Q) also sends the first and last query to its parent machine in the tree, i.e. sends the messages $(q_{\sigma_{(i-1)\cdot n_q+1}}, (i, Q), \text{first})$ and $(q_{\sigma_{i\cdot n_q}}, (i, Q), \text{last})$. After the parent node collects all the messages from the leaf node, it then does the same as its child: if it contains the start and end of a certain query q, it sends to the location information (the first and last machine that store it) of the query to its corresponding "meta-info" machine $(h(q), h_q)$, and it sends the first and last query and their location information to its parent machine. This is done recursively, and since there are $\lceil mN/s \rceil$ query data structure machines in total, the depth of this $\frac{s}{m}$-ary tree is $O(\log_{s/m} mN/s) = O(\frac{1}{\varepsilon'-\varepsilon})$, which means $O(\frac{1}{\varepsilon'-\varepsilon})$ rounds and $O(mN/s)$ machines suffice.

We do the same for (k, v) pairs. The sorting data structure machines (i, KV) sort the (k, v) pairs based on the order of k. As before, we build a $\frac{s}{2m}$-ary tree to send the location information to the "meta-info" machine of each key. The different part from query is that we combine the values that have the same key. For each machine in the $\frac{s}{2m}$-ary tree, it computes the averaged value associated with each key it contains and sends the averaged value to the corresponding "meta-info" machine. In particular, for each k_i, the 'meta-info" machine for k_i, $(h(k_i), h_k)$, contains the information $(k_i, \bar{v})$ where $\bar{v}$ is the average of v_j's such that $k_j = k_i$.

Next, the "meta-info" machines of query and key need to exchange information to retrieve the corresponding value for each query. Each (i, h_k) sends the $(k, \bar{v})$ pairs it has to the machine (i, h_q). Then, each (i, h_q) machine matches the q and k, and sends the associated value $\bar{v}$ to the q. Note that this step can be done by back propagating the $\frac{s}{m}$-ary tree constructed for sending the location information of q to $(h(q), h_q)$. In other words, we can just reverse the message sending direction in this tree. Therefore, each query in the query data structure machine receives the value it retrieves and from each query data structure machine (i, Q), we can send the retrieved value for each query to its corresponding token machine, which is the inverse of the second round.

To summarize, the total rounds needed is $O(\frac{1}{\varepsilon' - \varepsilon})$ and the number of machines needed is $O(mN/s) = O(N^{1+\varepsilon-\varepsilon'})$. To make this work for H heads, we can create H copies of this and each copy runs in parallel. Since $mH = O(N^{\varepsilon})$, the bounds for number of rounds and machines still hold. By creating this MPC simulation for each of the L-layers, we stack them in the order of layers yielding the complete simulation for L-layer EMA-transformer. □

Next, we generalize the above algorithm and proceed to simulate the ANN attention that can be computed by Algorithm 1. Since Algorithm 1 is a randomized algorithm, we assume that the MPC protocol shares all the random seeds needed for all the layers of ANNA-transformer.

Theorem C.3 (MPC simulates ANNA)**.** *Fix constants $0 < \varepsilon < \varepsilon' < 1$. For any L-layer ANNA-transformer T (as implemented by Algorithm 1) with $mH = O(N^{\varepsilon})$, there exists a $O(L/(\varepsilon' - \varepsilon))$-round MPC protocol π with local memory $s = O(N^{\varepsilon'})$ and $P = O(N^{1+\varepsilon-\varepsilon'+3/c^2})$ machines such that $\pi(\texttt{input}) = T(\texttt{input})$ for all $\texttt{input} \in \mathbb{Z}_{2^p}^N$.*

Proof. The high level idea of simulating ANNA-transformer is very similar to simulating EMA. We have the same kinds of machines as before. The biggest difference is that, instead of having one hash table for queries and keys, we now have ℓ hash tables, one for each round of hashing, and we sort the queries and keys based on the hash values of queries and keys. Again, we first outline different types of machines we will use.

- For $i \in [N]$, machine i is the *token machine* for x_i. This machine performs all the element-wise computation for token i. Specifically, it computes the query, key, value embeddings q_i, k_i, v_i and element-wise operations after the attention layer.
- For $i \in [\lceil mN/s \rceil], t \in [\ell]$, machine (i, Q, h^t) is a data structure machine for sorted queries for the t-th hash table and the i-th chunk of the sorted list of queries, where the ordering of sorting is based on $g_t(q)$ from Algorithm 1.
- For $i \in \lceil 2mN/s \rceil$, Machine (i, KV, h^t) is a data structure machine for sorted list of key and value pairs for the t-th hash table and the i-th chunk of the sorted list of key and value pairs, where the ordering of sorting is based on $g_t(k)$.
- For $i \in [N], t \in [\ell]$, Machine $(g_t(q_i), h_q, t)$ is the "meta-info" machine for the queries whose t-th hash value is $g_t(q_i)$. Let $h_{q_i}^t = \{q_j | j \in [N], g_t(q_j) = g_t(q_i)\}$. This machine stores the location information of $q \in h_{q_i}^t$ in the t-th hash table. Specifically, for all $q \in h_{q_i}^t$, this machine stores the start machine index, i.e. (start, Q, h^t) where $\text{start} = \arg\min_j \{q \in \text{Machine } (j, Q, h^t)\}$, and the end machine index, i.e. (end, Q, h^t) where $\text{end} = \arg\max_j \{q \in \text{Machine } (j, Q, h^t)\}$.
- For $i \in [N], t \in [\ell]$, Machine $(g_t(k_i), h_k, t)$ is the "meta-info" machine for the keys whose t-th hash value is $g_t(k_i)$. Let $h_{k_i}^t = \{k_j | j \in [N], g_t(k_j) = g_t(k_i)\}$. This machine stores the location information of $k \in h_{k_i}^t$ in the t-th hash table. Specifically, for all $k \in h_{k_i}^t$, this machine stores the start machine index, i.e. (start, KV, h^t) where $\text{start} = \arg\min_j \{k \in \text{Machine } (j, KV, h^t)\}$, and the end machine index, i.e. (end, Q, h^t) where $\text{end} = \arg\max_j \{k \in \text{Machine } (j, KV, h^t)\}$.
- The auxiliary machines needed for message propagation.

Like before, we still use each token machine to compute the embeddings $q_i, k_i, v_i \in \mathbb{R}^m$. Then, each token machine need to send (q_i, i) and (k_i, v_i, i) to the data structure machines, machine $(\lceil mi/s \rceil, Q, h^t)$ and machine $(\lceil 2mi/s \rceil, KV, h^t)$, for all $t \in \ell$. Because $\ell = N^{3\rho}$, we use the $\frac{s}{m}$-ary

tree to propagate the queries and keys to the corresponding data structure machines. This takes $O(\frac{1}{\varepsilon'-\varepsilon})$ rounds and $O(N^{1+3\rho+\varepsilon-\varepsilon'})$ machines.

Then, for each query hash table $t \in [\ell]$, the data structure machines sort the queries based on the hash value of the queries. Same as Theorem C.1, we use the $\frac{s}{m}$-ary tree to send the location information of each hash bucket to its corresponding "meta-info" machine. For each key, value pair hash table $t \in [\ell]$, the data structure machines sort the key, value pairs based on the hash value of the keys. After that, use the $\frac{s}{2m}$-ary tree to propagate the information to the corresponding "meta-info" machine. The difference from the EMA simulation is that each machine in this $\frac{s}{2m}$-ary tree maintains the sum of values whose key has the same hash values instead of the averaged value, and also maintains a count of the number of keys. These can be done in $O(\frac{1}{\varepsilon'-\varepsilon})$ rounds and $O(N^{1+3\rho+\varepsilon-\varepsilon'})$ number of machines.

Next, the key "meta-info" machine send the sum of values and count to the corresponding query "meta-info" machines, i.e. machine $(g_t(k_i), h_k, t)$ sends to machine $(g_t(q_i), h_q, t)$. Each query "meta-info" machine then follows the $\frac{s}{m}$-ary tree, broadcasting the sum of values and counts to the queries in the hash table. finally, each query in the hash table needs to propagate the information back to its original token machine. Since each token machine will receive message from $\ell = N^{3\rho}$ machines, we again reverse the $\frac{s}{m}$-ary tree that send the query to each data structure machine. During the aggregation, each machine in the $\frac{s}{m}$-ary tree still maintains the sum of values and the sum of counts it receives. After receiving the sum of values and counts, each token machine i then calculates $\text{ANNA}(q_i) =$ sum of the values divided by the counts.

The above simulates one layer of ANNA-transformer in $O(\frac{1}{\varepsilon'-\varepsilon})$ rounds and using $O(N^{1+3\rho+\varepsilon-\varepsilon'})$ machines, where $\rho = 1/c^2$. Therefore, by stacking the simulation for L layers, this gives $O(\frac{L}{\varepsilon'-\varepsilon})$ rounds in total. To extend to H heads, we just need to instantiate the above simulation for H parallel copies and because $mH = O(\varepsilon)$, the total number of rounds and machines still remains the same. □

D ANN/EM Attention can simulate low-rank Attention via MPC

We simulate the low-rank attention using ANN attention by first giving a MPC algorithm for computing low-rank attention and then convert it to ANNA-transformer.

Theorem D.1 (ANNA/EMA simulates low-rank Attention)**.** *For constants $0 < \varepsilon < \varepsilon' < 1$, any low-rank attention based transformer with depth L, rank r, embedding dimension m and $rm = O(N^{\varepsilon})$ can be simulated by an EMA/ANNA-transformer with depth $O(\frac{L}{\varepsilon'-\varepsilon})$, number of heads $H = O(N^{\varepsilon'})$ and embedding dimension $m = O(N^{5\varepsilon'} \log N)$.*

Proof. We prove this theorem by first proving that any one-layer of low-rank attention can simulated by constant number of rounds of MPC.

Lemma D.2 (MPC simulates low-rank Attention)**.** *For constants $0 < \varepsilon < \varepsilon' < 1$, any one-layer low-rank attention with rank r, embedding dimension m and $rm = O(N^{\varepsilon})$ can be simulated by a $O(\frac{1}{\varepsilon'-\varepsilon})$-round MPC protocol with local memory $s = O(N^{\varepsilon'})$ and $O(N)$ machines.*

Proof. Assume $rm = O(N^{\varepsilon})$ and local memory of MPC $s = O(N^{\varepsilon'})$ where $\varepsilon < \varepsilon'$. Same as what we do in MPC simulating EMA, for each token $x_i, i \in [N]$, we have a token machine i to compute the embedding of x_i but we need to compute it in the kernel space, i.e. $q_i = Q'(x_i), k_i = K'(x_i)$ and $v_i = V(x_i)$. To compute $K'(X)^{\mathsf{T}}V(X)$, recall that

$$K'(X)^{\mathsf{T}}V(X) = \sum_{i=1}^{N} k_i v_i^{\mathsf{T}}$$

We just need to compute the sum of N matrices of size $r \times m$. Each token machine i computes the matrix $k_i v_i^{\mathsf{T}}$ and we construct a $\lfloor \frac{s}{rm} \rfloor$-ary tree of machines to compute the sum. The leaves of the tree are all the token machines and each node is responsible for computing the sum of $\lfloor \frac{s}{rm} \rfloor$ number of matrices. We know from the previous simulation that the depth of the tree is $O(\frac{1}{\varepsilon'-\varepsilon})$. After we obtain the matrix $M = K'(X)^{\mathsf{T}}V(X) \in \mathbb{R}^{r \times m}$, in order to compute $Q(X)K'(X)^{\mathsf{T}}V(X)$, we just

need to propagate the matrix M to all the token machines. And each token machine i computes $q_i^\top M$. By reversing the direction of message propagation in the computing sum tree, we can propagate M to all the token machines in $O(\frac{1}{\varepsilon'-\varepsilon})$ rounds. Therefore, we can simulate kernel attention with $O(\frac{1}{\varepsilon'-\varepsilon})$ rounds in total. □

For L layers of low-rank attention transformer, we construct the MPC for each layer using Lemma D.2 and again we use the local computation of each token machine to simulate the element-wise computation. We stack the L MPCs together, which has $O(\frac{L}{\varepsilon'-\varepsilon})$ rounds. The theorem follows from applying Theorem B.2 and Corollary B.9. □

Since the core of the proof is through MPC simulating low-rank Attention, we can also apply Theorem B.10 and Corollary B.14 which simulate MPC with better embedding dimension to get a improved embedding dimension for simulating low-rank attention transformer.

Corollary D.3 (ANNA/EMA simulates low-rank Attention with improved embedding dimension)**.** *For constants $0 < \varepsilon < \varepsilon' < 1$, any low-rank attention based transformer with depth L, rank r, embedding dimension m and $rm = O(N^\varepsilon)$ can be simulated by an EMA/ANNA-transformer with depth $O(\frac{L}{(\varepsilon'-\varepsilon)\cdot\min(\varepsilon^2,(\varepsilon'-\varepsilon)^2)})$, number of heads $H = O(N^{\frac{\varepsilon'-\varepsilon}{4}})$ and embedding dimension $m = O(N^{\varepsilon'})$.*

E Discussion on Reformer

We formally define Reformer as a computational model here.

Definition E.1 (Reformer attention)**.** *Given query, key, value embeddings $Q(X), K(X), V(X) \in \mathbb{R}^{N\times m}$ such that $q_i := k_i = Q(X)[i,:] = K(X)[i,:], v_i = V(X)[i,:]$,* Reformer *attention proceeds as follows:*

1. *Apply a hash function $h : \mathbb{R}^m \to U$ on $\{q_1, \ldots, q_N\}$;*
2. *Sort all q_i's (and thus k_i's) by $h(q_i)$ and partition all q_i's into chunks of size $B \le O(1)$, and let $h'(q_i)$ be the label of the chunk that q_i is in (the queries in each chunk can have different hash values);*
3. *For each q_i, only attend to k_j's such that they are in the same chunk.*

The output embedding for q_i is therefore

$$\sum_{j:h'(k_j)=h'(q_i)} \frac{\exp(\langle q_i, k_j\rangle)}{\sum_{j':h'(k_{j'})=h'(q_i)} \exp(\langle q_i, k_{j'}\rangle)} \cdot v_j.$$

We define $f_\ell : [N] \to [N]^B$ as the function that specifies the set of keys each query should compute inner product with in the ℓ-th layer. From the Reformer constraints, we have $\forall i \in [N]$:

1. $f_\ell(i) = \{a_1, a_2, \ldots, a_B\} \in [N]^B$ is a set (no repetition).
2. $i \in f_\ell(i)$.
3. For any $j \in f_\ell(i)$, $f_\ell(j) = f_\ell(i)$.

In the ℓ-th layer attention computation for each query q_i, Reformer computes

$$\sum_{j\in f_\ell(i)} \frac{\exp(\langle q_i, k_j\rangle)}{\sum_{j'\in f_\ell(i)} \exp(\langle q_i, k_{j'}\rangle)} \cdot v_j.$$

We first study a restricted version of Reformer that fix the communication pattern beforehand i.e. f_ℓ is input-independent for all $\ell \in [L]$, and show that it can not compute the sum of all the input tokens.

Definition E.2 (SUM). *Given input $X = (x_1, x_2, \ldots, x_N), x_i \in [M]$, and $M = N^{O(1)}$, the* SUM *task is defined as* $\text{SUM}(X) = \sum_{i=1}^{N} x_i$. *We say a Reformer T computes* SUM *if for all X, $T(X)_N = \text{SUM}(X)$.*

Here $T(X)_N$ is the N-th output of T given the input X. One can choose any position to be the final output position, and here WLOG we choose the last token to follow the autoregressive generation model convention.

Proposition E.3. *Fix $L = O(1)$ and $\{f_\ell\}_{\ell=1}^{L}$. Any Reformer T with L layers and each layer the attention pattern is specified by $\{f_\ell\}_{\ell=1}^{L}$ can not compute* $\text{SUM}(X)$*: there exists an X, $|T(X)_N - \text{SUM}(X)| \geq \epsilon$, for any $0 < \epsilon < M/2$.*

Proof. We denote each layer's element-wise computation by $\{\phi_\ell\}_{\ell=1}^{L}$. Let $T^\ell(X)_i$ denote the i-th output of T after ℓ layers of computation. We prove this proposition by induction.

Inductive hypothesis: $T^\ell(X)_i$ is a function of at most B^ℓ different $x_i \in X$.

Base case: $\ell = 1$

$$\begin{aligned} T^1(X)_i &= \sum_{j \in f_1(i)} \frac{\exp(\langle q_i, k_j \rangle)}{\sum_{j' \in f_1(i)} \exp(\langle q_i, k_{j'} \rangle)} \cdot v_j \\ &= \sum_{j \in f_1(i)} \frac{\exp(\langle Q(x_i), K(x_j) \rangle)}{\sum_{j' \in f_1(i)} \exp(\langle Q(x_i), K(x_{j'}) \rangle)} \cdot V(x_j) \\ &= \phi_1(x_{a_1}, x_{a_2}, \ldots, x_{a_B}) \text{ where} f_1(i) = \{a_1, \ldots, a_B\} \end{aligned}$$

which is a function of at most B x_i's in X.

Inductive step: consider

$$\begin{aligned} T^{\ell+1}(X)_i &= \sum_{j \in f_{\ell+1}(i)} \frac{\exp(\langle Q(T^\ell(X)_i), K(T^\ell(X)_j) \rangle)}{\sum_{j' \in f_{\ell+1}(i)} \exp(\langle Q(T^\ell(X)_i), K(T^\ell(X)_{j'}) \rangle)} \cdot V(T^\ell(X)_j) \\ &= \phi_{\ell+1}(T^\ell(X)_{a_1}, \ldots, T^\ell(X)_{a_B}) \text{ where} f_1(i) = \{a_1, \ldots, a_B\} \end{aligned}$$

Since each $T^\ell(X)_{a_j}$ is a function of at most B^ℓ variables from X, $T^{\ell+1}(X)_i$ is a function of at most $B \cdot B^\ell = B^{\ell+1}$ variables from X.

Therefore, if $T^L(X)_i$ is a function of all $\{x_1, \ldots, x_N\}$, we need $B^L \geq N$ and thus $L = \Omega(\log_B N)$. In the case $B = O(1)$ and $L = O(1)$, $T^L(X)_i$ is a function of $B^L \ll N$ variables. WLOG, consider $T^L(X)_N$ is a function of $\{x_1, \ldots, x_{B^L}\}$. Then, x_{B^L+1} can be any number in $[M]$ that makes $T^L(X)_N$ far from $\text{SUM}(X)$. □

Therefore, if Reformer has any power, it must come from the sorting part, because the sorting algorithm have access to the information of all the token inputs.

Although constant-layer Reformer can not compute SUM, one can easily show that one layer of ANNA-transformer can compute SUM by setting $v_i = Nx_i$ and $k_1 = k_2 = \cdots = k_N = q_N$ for all $i \in [N]$, thereby retrieving all the v_i's and averaging them.

F ANNA-transformer Solves k-hop and Match2

F.1 ANNA/EMA-transformer Solves Match2

Theorem F.1. *For any $N, M = N^{O(1)}$, there exists an EMA-transformer T with one layer, one attention head, and embedding dimension 1 such that $T(X) = \text{Match2}(X)$ for all $X \in [M]^N$.*

Proof. Given input $X \in [0, M]^{N\times 1}$. Let $Q(X) = \phi(X)Q, K(X) = \phi(X)K, V(X) = \phi(X)V$, where Q, K, V are matrices in $\mathbb{R}^{2\times 1}$. Define ϕ by $\phi(x) = (x, 1)$ and

$$Q = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, K = \begin{pmatrix} -1 \\ M \end{pmatrix}, V = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

such that

$$\phi(X)Q = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}, \phi(X)K = \begin{pmatrix} M - x_1 \\ M - x_2 \\ \vdots \\ M - x_N \end{pmatrix}, \phi(X)V = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}.$$

As a result, for each $1 \le i \le N$, if there exists $1 \le j \le N$ such that $x_i + x_j = M$, then

$$((\phi(X)Q)(\phi(X)K)^\intercal)[i, j] = \frac{1}{|\{j \in [N] : x_i + x_j = M\}|}.$$

Otherwise, $((\phi(X)Q)(\phi(X)K)^\intercal)[i, j] = 0$ for all $1 \le j \le N$. Finally, we can calculate that if $|\{j \in [N] : x_i + x_j = M\}| \neq 0$, then

$$\text{EMA}(\phi(X)Q, \phi(X)K, \phi(X)V)[i] = \frac{1}{|\{j \in [N] : x_i + x_j = M\}|} \cdot |\{j \in [N] : x_i + x_j = M\}| = 1,$$

and if $|\{j \in [N] : x_i + x_j = M\}| = 0$, then

$$\text{EMA}(\phi(X)Q, \phi(X)K, \phi(X)V)[i] = 0. \qquad \square$$

That gives the same result for ANNA-transformers.

Corollary F.2. *For any $N, M = N^{O(1)}$, there exists an ANNA-transformer T with one layer, one attention head, and embedding dimension 1 such that $T(X) = \text{Match2}(X)$ for all $X \in [M]^N$.*

F.2 ANN transformer Solves k-hop

We first show that ANNA transformers can solve induction head (1-hop).

Lemma F.3. *Fix constants $0 < \varepsilon < \varepsilon' < 1$, and $|\Sigma| \le N$. There exists an ANNA-transformer T with $L = O(\frac{1}{\varepsilon\cdot(\varepsilon'-\varepsilon)^2})$ layers, $H = O(N^{(\varepsilon'-\varepsilon)/4})$ heads per layer, and embedding dimension $m = O(N^{\varepsilon'})$ such that $T(w)_i = w_{\sigma(w,i)}$, if $\sigma(w, i) \neq 0$; $T(w)_i = \perp$, if $\sigma(w, i) = 0\ \forall i \in [N]$, for all $w \in \Sigma^N$.*

Proof. We prove this lemma by designing a constant-round MPC algorithm with local memory $s = O(N^\varepsilon)$ and N/s machines to solve 1-hop. Since $|\Sigma| \le N$, each token can be embedded with $O(\log N)$ bits ($O(1)$ words). Denote the input $w^N = (x_1, x_2, \ldots, x_N)$. The MPC algorithm works as following:

1. For each x_i, retrieve the next token x_{i+1} and each token on the machine is stored as the embedding of $(x_i, i, x_{i+1}, i + 1)$.
2. Define a comparator $<$ for the object $(x_i, i, x_{i+1}, i + 1)$. For two tuples $(x_i, i, x_{i+1}, i + 1)$ and $(x_j, j, x_{j+1}, j + 1)$, if $x_i \neq x_j$, then $x_i < x_j \Rightarrow (x_i, i, x_{i+1}, i + 1) < (x_j, j, x_{j+1}, j + 1)$; if $x_i = x_j$, then $i < j \Rightarrow (x_i, i, x_{i+1}, i + 1) < (x_j, j, x_{j+1}, j + 1)$. Sort $(x_i, i, x_{i+1}, i + 1)$ by the comparator $<$.
3. Each token $(x_i, i, x_{i+1}, i + 1)$ in the sorted list retrieves the token before it in the sorted list, denoted by $(x_j, j, x_{j+1}, j + 1)$. Update the embedding of token: if $x_j = x_i$, the embedding of the token x_i becomes $(i, x_{j+1}, j + 1)$ i.e. $(i, w_{\sigma(w,i)}, \sigma(w, i))$; if $w_j \neq w_i$, then the embedding of the token x_i becomes $(i, \perp, 0)$.
4. Send each $(i, w_{\sigma(w,i)}, \sigma(w, i))$ to the correct output machine $\lceil \frac{i}{s} \rceil$ and output $w_{\sigma(w,i)}$ for token i.

For step 1, each machine only needs to send message to its neighbor machine: machine i sends message to machine $i-1$, and this only takes 1 round. In step 2, each tuple is only $O(\log N)$ bits, so by Lemma C.2, the sorting takes $O(\frac{1}{\varepsilon})$ rounds. In step 3, again each machine only needs to send message to its neighbor machine: machine i sends message to machine $i+1$, and this only takes 1 round. In step 4, each machine for the sorted list sends at most s tuples stored in it to the correct output machine which takes 1 round. Thus, the MPC algorithm has $O(\frac{1}{\varepsilon})$ rounds in total.

Then, we convert this MPC algorithm to an ANNA-transformer. By Corollary B.14, this gives us an ANNA-transformer with number of heads $H = O(N^{(\varepsilon'-\varepsilon)/4})$, embedding dimension $m = O(N^{\varepsilon'})$ and number of layers $L = O(\frac{1}{\varepsilon\cdot(\varepsilon'-\varepsilon)^2})$. □

Now we show that ANNA-transformers can solve k-hop with $O(\log k)$ layers.

Theorem F.4. *Fix constants $0 < \varepsilon < \varepsilon' < 1$, $|\Sigma| \le N$ and any $k \in \mathbb{N}$. There exists an ANNA-transformer T with $L = O\left(\frac{1}{\varepsilon\cdot(\varepsilon'-\varepsilon)^2} + \frac{\log k}{(\varepsilon'-\varepsilon)^2}\right)$ layers, $H = O(N^{(\varepsilon'-\varepsilon)/4})$ heads per layer, and embedding dimension $m = O(N^{\varepsilon'})$ such that $T(w)_i = w_{\sigma^k(w,i)}, \forall i \in [N]$, for all $w \in \Sigma^N$.*

Proof. We prove this theorem by constructing an $O(\log k)$-rounds MPC with $s = O(N^{\varepsilon})$ local memory and $O(N/s)$ machines. We show that this MPC algorithm can compute k-hop by induction. Let $k = \sum_{j=0}^{\lfloor \log k \rfloor} k_j 2^j$ and $k_{:\ell} = \sum_{j=0}^{\ell-1} k_j 2^j$, where $k_j \in \{0,1\}$.

Inductive hypothesis: after $O(\frac{1}{\varepsilon}) + 2\ell$ rounds of MPC computation, the token embedding for each token i encodes the information of this tuple

$$(i, w_{\sigma^{2^\ell}(w,i)}, \sigma^{2^\ell}(w,i), w_{\sigma^{k_{:\ell}}(w,i)}, \sigma^{k_{:\ell}}(w,i))$$

Base case: $\ell = 0, k = 1$ is implied by Lemma F.3. After step 3, we have $(i, w_{\sigma(w,i)}, \sigma(w,i))$. Now consider $k = \ell + 1$. For each i, the machine (Machine $\lceil i/s \rceil$) that contains $(i, w_{\sigma^{2^\ell}(w,i)}, \sigma^{2^\ell}(w,i), w_{\sigma^{k_{:\ell}}(w,i)}, \sigma^{k_{:\ell}}(w,i))$ sends the message $(i, \sigma^{2^\ell}(w,i))$ to machine that contains $\sigma^{2^\ell}(w,i)$ as the first entry of the tuple which is machine $\lceil \frac{\sigma^{2^\ell}(w,i)}{s} \rceil$. Machine $\lceil \frac{\sigma^{2^\ell}(w,i)}{s} \rceil$ then send the following tuple to machine $\lceil i/s \rceil$:

$$\begin{aligned}
&(\sigma^{2^\ell}(w,i), w_{\sigma^{2^\ell}(w,\sigma^{2^\ell}(w,i))}, \sigma^{2^\ell}(w,\sigma^{2^\ell}(w,i)), w_{\sigma^{k_{:\ell}}(w,\sigma^{2^\ell}(w,i))}, \sigma^{k_{:\ell}}(w,\sigma^{2^\ell}(w,i)))\\
&= (\sigma^{2^\ell}(w,i), w_{\sigma^{2^{\ell+1}}(w,i)}, \sigma^{2^{\ell+1}}(w,i), w_{\sigma^{k_{:\ell}+2^\ell}(w,i)}, \sigma^{k_{:\ell}+2^\ell}(w,i))\\
&:= (\sigma^{2^\ell}(w,i), t_1, t_2, t_3, t_4)
\end{aligned}$$

Since each machine has at most s tuples and the function $\sigma(w,i)$ is one-to-one except for $\perp$, the number of messages each machine sends and receive is bounded by s. After machine $\lceil i/s \rceil$ receiving the above message, it update the tuple for the token i:

1. if $k_\ell = 0$, token i is updated as: $(i, t_1, t_2, w_{\sigma^{k_{:\ell}}(w,i)}, \sigma^{k_{:\ell}}(w,i))$
2. if $k_\ell = 1$, token i is updated as: (i, t_1, t_2, t_3, t_4)

By definition, the embedding of token i now is:

$$(i, w_{\sigma^{2^{\ell+1}}(w,i)}, \sigma^{2^{\ell+1}}(w,i), w_{\sigma^{k_{:\ell+1}}(w,i)}, \sigma^{k_{:\ell+1}}(w,i))$$

The above inductive step only takes 2 rounds of MPC. Therefore, the total round is $O(\frac{1}{\varepsilon}) + 2(\ell+1)$. When $\ell = \lfloor \log k \rfloor + 1$, this algorithm compute the output for k-hop.

Again, we can convert this MPC algorithm to an ANNA-transformer. By Corollary B.14, this gives us an ANNA-transformer with number of heads $H = O(N^{(\varepsilon'-\varepsilon)/4})$, embedding dimension $m = O(N^{\varepsilon'})$ and number of layers $L = O(\frac{1}{\varepsilon\cdot(\varepsilon'-\varepsilon)^2} + \frac{\log k}{(\varepsilon'-\varepsilon)^2})$. □

G Experimental details

Here are the details of the experimental setup. All the experiments are launched on 2 GPUs: NIVIDIA Titan RTX and NVIDIA Titan Xp.

We train a modified version of the attention matrix and then distill from the trained model using ANNA implemented by the angular distance LSH family from [6]. Our softmax attention normalizes all the queries and keys in $Q(X)$ and $K(X)$ to have unit norm, and computes $\text{softmax}(\beta \cdot Q(X)K(X)^\top)V(X)$ with a hyperparameter $\beta > 0$.

G.1 Match2 experiments

Dataset generation. Inspired by the way [37] generating data for Match3 task, a triple-wise version of Match2, we generate the data for Match2 using the same algorithm but change to pair-wise relation when computing the label. Each sample is a tuple (X, Y), where $X = (x_1, x_2, \ldots, x_N)$, and each x_i is an integer sampled from $\{1, 2, \ldots, 36\}$; $Y = (y_1, y_2, \ldots, y_N)$, and each $y_i = \mathbb{1}\{\exists j \centerdot x_i + x_j = 0 \bmod 37\}$. The sequence length N is set to 32. When sampling the data, we ensure that each batch is balanced by having the distribution of one's in Y the same: each batch has 4 bins and each bin corresponds to each percentage $[0, 25\%)$, $[25\%, 50\%)$, $[50\%, 75\%)$, $[75\%, 100\%]$ of one's in Y; each bin size is $1/4$ of the batch size. See Algorithm 3 for details.

Algorithm 3 Match2 Dataset Generation

Input: $N = 32$, $M = 37$, dataset size D
Output: Dataset of (X, Y) pairs

Initialize 4 empty bins for ones-percentage ranges: $[0, 25)$, $[25, 50)$, $[50, 75)$, $[75, 100]$
$N_b \leftarrow D/4$
while total examples in bins $< D/40$ **do**
    Sample $X \in \{1, \ldots, M\}^N$ uniformly at random
    Calculate the percentage p of one's in Y
    if size of the bin p is in $< N_b$ **then**
        Add (X, Y) to the correct bin
for each bin **do**
    while size of bin $< N_b$ **do**
        Randomly sample (X, Y) from bin
        Sample permutation π over $[0, \ldots, N-1]$
        $X^* \leftarrow X[\pi]$, $Y^* \leftarrow Y[\pi]$
        Add (X^*, Y^*) to bin
Combine and shuffle all bins into final dataset
return Dataset

Training details. We trained 3 models with $\beta \in \{0.1, 1, 10\}$ respectively, with Adam optimizer on cross-entropy loss and learning rate 0.01. Each model has one layer, one attention head, embedding dimension $m = 64$ and an MLP with width $4m$ and GeLU activation. The dataset size, batch size, training steps are $10000, 32, 20000$ respectively.

We apply ANNA with number of hash tables $\ell \in \{1, 2, \ldots, 16\}$ and number of hash functions for each table $z \in \{1, 2, \ldots, 6\}$ on all the 3 trained models, and $\beta = 0.1$ has the best performance (error can be 0). Since the implementation of ANNA is randomized, for each combination of (ℓ, z), we run 10 times and report the averaged error over the 10 runs. See Figure 1a for plotted performance when $\beta = 0.1$. In this setting, $\ell \geq 8, z = 1$ can achieve 0 error on the test set with 256 test samples.

G.2 Induction heads experiments

Dataset generation. We use the data generation algorithm from [55]. Each sample (X, Y) is of the form $X = (k, X')$ for $k \in \{0, 1\}$ for training samples and $k = 1$ for test samples, $X' \in \Sigma^{N-1}$, $Y = (0, Y')$, where Y' is the k-hop label for X'. Here the sequence length $N = 100$ and alphabet size $|\Sigma| = 4$.

Training details. We trained 3 models with $\beta \in \{0.1, 1, 10\}$ respectively, with Adam optimizer on cross-entropy loss and learning rate 0.01. Each model has 2 layers, each layer with one attention head, embedding dimension $m = 128$ and an MLP with width $4m$ and GeLU activation. We use online training: at each training step, sample fresh new data to train. The batch size and training steps are $32, 400000$ respectively.

We apply ANNA on all the 3 trained models. For the first layer, the number of hash tables ℓ is chosen from $\{32, 40, 48, \ldots, 96\}$ and z is chosen from $\{1, 2, 3, 4\}$. For the second layer, the number of hash tables ℓ is chosen from $\{4, 8, 12, \ldots, 32\}$ and z is chosen from $\{1, 2, 3, 4\}$. When evaluating the test error, we compute the error on all the tokens. Note that this is different from [55], where they only compute the error on the tokens whose induction head exists to avoid overestimating the performance when k is large and has a large fraction of null outputs. In our setting, $k = 1$ which doesn't have many null outputs, and it is important for the model to learn when to output the null token.

We found $\beta = 1$ has the best performance, so we report it in Figure 1b. Again, the errors are averaged over 10 runs for each combinations and taken the minimum over z's. One can see that 32 hash tables in the first layer and 4 hash tables in the second layer already gives highly non-trivial performance: the error rate is 0.2 over 100 samples and each sample has 100 token predictions, while random guess would give 0.75 error rate. With more hash tables in the first layer, the error rate can go below 0.1.