

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 $\eta = O(1/N^{1-3\rho})$.*

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 probability greater than $1 - \frac{0.1}{N}$, there is no collision in one hash table. By [29], $z = O(\log_{1/p_2} N)$, 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, $\text{count} \leq (\mathcal{N}(q_i) - 1) \cdot \ell$, and this gives us $w_{i,j} \geq \frac{1}{(\mathcal{N}(q_i) - 1) \cdot \ell + 1}$. \square

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.

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.

¹Optimal (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 .

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

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

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

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

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

970 *Proof.* For any R -round MPC protocol π with N machines that maps the input to output, we define
 971 the intermediate steps for local computation phase and message transmission phase. We denote the in-
 972 put to all the machines before the local computation as $\text{MachineIn}_1, \text{MachineIn}_2, \dots, \text{MachineIn}_R$,
 973 and denote the information after deterministic local computations $(\text{Local}_r^i)_{r \in [R], i \in [N]}$ as
 974 $\text{MachineOut}_1, \text{MachineOut}_2, \dots, \text{MachineOut}_R$, where $\text{MachineOut}_r^i = \text{Local}_r^i(\text{MachineIn}_r^i)$.
 975 In the communication (message transmission) phase, we need to route the messages to the correct
 976 machines ie from MachineOut_r to MachineIn_{r+1} .

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

- 981 1. Initialization. The input feeded into EMA-transformer is distributed in the N tokens, and we
 982 need to transfer them into the first $\lceil \frac{N}{s} \rceil$ tokens/machines to match MachineIn_1 .
- 983 2. Routing (message transmission). After the local computation in each round r , we need to
 984 communicate the messages from MachineOut_r to MachineIn_{r+1} .
- 985 3. Final output. The MPC output is distributed in the first $\lceil \frac{N}{s} \rceil$ tokens/machines, and we need to
 986 distributed them back to the N tokens.

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

988 We first show the message transmission part of MPC can be simulate by the EMA-transformer. Recall
 989 that after r rounds of local computation, each machine i has a set of messages it wants to send to
 990 other machines, denoted by $\text{MachineOut}_r^i = \{(\text{Msg}_{\text{dest}}^i, \text{dest}) : \text{dest} \in \text{sent}^i\}$, where sent^i is
 991 the set of machine indices that machine i will send the message to and $\text{Msg}_{\text{dest}}^i$ is the message machine
 992 i send to machine dest . After the message communication phase, each machine i has the set of
 993 messages it receives from other machines, denoted by $\text{MachineIn}_{r+1}^i = \{(\text{Msg}, \text{Src}) : (\text{Msg}, i) \in$
 994 $\text{MachineOut}_r^{\text{Src}}\}$. Since each machine can only send/receive s words, we have $\sum_{\text{dest} \in \text{sent}^i} |\text{Msg}| \leq$
 995 s and $\sum_{(\text{Msg}, i) \in \text{MachineOut}_r^{\text{Src}}} |\text{Msg}| \leq s$ for all machine i . We call this process the routing process of
 996 MPC. The following lemma shows that each routing round of MPC can be simulated by one layer of
 997 EMA-transformer.

998 **Lemma B.3** (Routing). For any R -round MPC protocol π having q machines each with local memory
 999 s and any $r \in [R - 1]$, there exists an EMA-transformer route_r with $H = O(s)$ heads and
 1000 $m = O(\log q)$ for Q and K , $m = O(s^5 \log q)$ for V that takes input $X = \text{MachineIn}_r$ and produces
 1001 output $\text{route}_r(X) = \text{MachineIn}_{r+1}$

1002 *Proof.* Follow the assumption in [53], we encode the local computation into the element-wise
 1003 operations $Q(\cdot), K(\cdot), V(\cdot)$ of transformer. The main part of the proof will focus on using EMA to
 1004 route MachineOut_r to MachineIn_{r+1} .

1005 We assign a unique positional encoding or identifier to each machine i , denoted by p_i . This can
 1006 be done with $O(\log q)$ bits. This encoding serves as a unique key to retrieve the message in each
 1007 machine. The high level idea is to create a query for each machine i and a key for each $\text{dest} \in \text{sent}_i$
 1008 and the associated value is the message $\text{Msg}_{\text{dest}}^i$ sent to dest in the protocol. Since each machine can
 1009 send at most s messages to other machines, we create s EMA heads and each head is responsible for
 1010 one message for all the q machines. Each machine retrieves the message sent to them by having a
 1011 query in each head. Because each query can attend only to perfectly matching keys, each distinct
 1012 outbound message must be passed by a different attention head, but multiple inbound messages may
 1013 be received by the same attention head.

1014 Specifically, let Q^h, K^h, V^h be the query, key, value embedding after the machine local computation
 1015 for each head $h \in [s]$. Set $q_i^h = p_i$ for all h , so

$$Q^1 = Q^2 = \dots = Q^s = \begin{pmatrix} p_1^\top \\ p_2^\top \\ \vdots \\ p_q^\top \end{pmatrix}.$$

1016 Let $k_i^h = p_{\text{dest}_h^i}$ for $\text{dest}_h^i \in \text{sent}^i = \{\text{dest}_1^i, \text{dest}_2^i, \dots, \text{dest}_s^i\}$, where dest_j^i is the destination
 1017 machine index for the j th word message that machine i sends. The key matrices are constructed as
 1018 follows:

$$K^1 = \begin{pmatrix} p_{\text{dest}_1^1}^\top \\ p_{\text{dest}_2^1}^\top \\ \vdots \\ p_{\text{dest}_q^1}^\top \end{pmatrix}, \quad K^2 = \begin{pmatrix} p_{\text{dest}_1^2}^\top \\ p_{\text{dest}_2^2}^\top \\ \vdots \\ p_{\text{dest}_q^2}^\top \end{pmatrix}, \quad \dots, \quad K^s = \begin{pmatrix} p_{\text{dest}_1^s}^\top \\ p_{\text{dest}_2^s}^\top \\ \vdots \\ p_{\text{dest}_q^s}^\top \end{pmatrix}.$$

1019 Let v_i^h be some embedding of $(\text{Msg}_{\text{dest}_h^i}^i, \text{dest}_h^i, i)$, denoted by $v_i^h = \text{emb}_i^h(\text{Msg}_{\text{dest}_h^i}^i, \text{dest}_h^i, i)$ for
 1020 some emb_i^h defined later, and

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

1021 By such construction of Q, K, V , in our EMA, each query will retrieve the average value of the
 1022 messages whose key exactly matches the query. However, by setting the value matrix this way, we
 1023 might corrupt the message when there are more than one $k_i^h \in K^h$ that are equal to the same query.
 1024 To solve this problem, we can apply the same *multiple hashing-based encoding* in Lemma 3.2 from
 1025 [53], which encodes each message in multiple fixed locations generated by a sparse binary matrix
 1026 and have an extra “validity bit” indicating whether the message is corrupted or not. We restate an
 1027 adapted version of their Lemma 3.2 here.

1028 **Lemma B.4** (Lemma 3.2 of [53]; message encoding in sparse averaging). *For any message size*
 1029 $\Delta \in \mathbb{N}$, *message count bound* $\alpha \in \mathbb{N}$, *there exist an encoding function* ϕ *such that* ϕ *takes in*
 1030 $(\text{Msg}_{\text{dest}_h^i}^i, \text{dest}_h^i, i)$ *defined above where the size of it is bounded by* Δ *for all* $i \in [q]$ *and* $h \in [\alpha]$
 1031 *and encodes it into* $\text{emb}_i^h(\text{Msg}_{\text{dest}_h^i}^i, \text{dest}_h^i, i) \in \mathbb{R}^m$ *with* $m = O(\alpha^4 \Delta \log q)$, *and a decoder function*
 1032 φ *such that* φ *takes in the output of the EMA with* Q, K, V *defined above and decodes it into*
 1033 $(\text{Msg}_{\text{dest}_h^i}^i, \text{dest}_h^i, i)$.

1034 Let $\text{rcvd}^i = \{\text{src}_1^i, \text{src}_2^i, \dots, \text{src}_s^i\}$, where src_j^i is the j th source machine index that machine i
 1035 receives message from. Because $|\text{sent}^i| \leq s$ and $|\text{rcvd}^i| \leq s$, in each head of EMA, there are at
 1036 most s values get retrieved and averaged for each query. Thus, here we can just apply Lemma B.4
 1037 with $\alpha = \Delta = s$, which gives us an embedding dimension bound $m = O(s^5 \log q)$. \square

1038 We then show with one layer EMA-transformer, we can properly initialize the setup of MPC by
 1039 converting $\text{Input} = (\text{input}_1, \text{input}_2, \dots, \text{input}_n)$ to MachineIn_1 , the input before the first round
 1040 of MPC computation, ie the input is distributed evenly on the first $\lceil \frac{n}{s} \rceil$ machines.

1041 **Lemma B.5 (Initialization).** *For any R -round MPC protocol π having q machines each with local*
 1042 *memory s and n -word input, there exists an EMA-transformer init with $H = 1$ head, $m = O(\log q)$*
 1043 *for Q, K and $m = O(s)$ for V that takes input and outputs $\text{init}(\text{input}) = \text{MachineIn}_r$.*

1044 *Proof.* The input should be distributed accross each machine $1 \leq i \leq \lceil \frac{n}{s} \rceil$ with $\text{MachineIn}_1^i =$
 1045 $\{(\text{input}_{\text{idx}}, \text{idx}) : \text{idx} \in s(i-1)+1, \dots, \min\{n, si\}\}$. Let $q_{in} = \lceil \frac{n}{s} \rceil$ be the number of machines
 1046 to store the initial input. Since the input given to init are n tokens (here we treat each token as a
 1047 machine), we need to rearrange the memory so that the input is distributed on the first q_{in} tokens.

1048 Same as before, we use the positional encoding p_i to be the unique identifier for each machine. We
 1049 create a key value pair for each input token and the key corresponds to the identifier of the machine
 1050 that $\text{input}_{\text{idx}}$ goes to and the value be $(\text{input}_{\text{idx}}, \text{idx})$. Also, create a query for each machine
 1051 $i \in [q_{in}]$.

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

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

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

$$K = \begin{pmatrix} p_{\text{dest}_1}^\top \\ p_{\text{dest}_2}^\top \\ \vdots \\ p_{\text{dest}_n}^\top \end{pmatrix}$$

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

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

1052 By setting the value matrix like this, we can avoid corrupting the messages. □

1053 Last, we show that with an additional one layer EMA-transformer, we can map the final round
 1054 MachineIn_R to the output of MPC protocol where the output is store in the first $\lceil \frac{n}{s} \rceil$ machines.

1055 **Lemma B.6 (Final Output).** *For any R -round MPC protocol π having q machines each with local*
 1056 *memory s and n -word input, there exists an EMA-transformer out with $H = 1$ head, $m = O(\log q)$*
 1057 *for Q, K and $m = O(s)$ for V that takes MachineIn_R and outputs $\text{out}(\text{MachineIn}_R)_{:n} =$*
 1058 *$\pi(\text{input}) = \text{output}$.*

1059 *Proof.* First, the element-wise operations can compute MachineOut_R from MachineIn_R . The output
 1060 is distributed accross each machine $1 \leq i \leq \lceil \frac{n}{s} \rceil = q_{out}$ with memory of machine i be $\text{output}^i =$
 1061 $\{(\text{output}_{\text{idx}}, \text{idx}) : \text{idx} \in s(i-1)+1, \dots, \min\{n, si\}\}$. Then, we just need to retrieve the output
 1062 tokens from all the q_{out} machines and distribute them back to n tokens. This step does the inverse
 1063 job of init . We create a query for each token $\text{output}_{\text{idx}}$ for all $\text{idx} \in [n]$. Let $\text{src}_{\text{idx}} = \lceil \frac{\text{idx}}{s} \rceil$ be the
 1064 machine token $\text{output}_{\text{idx}}$ is in.

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

$$Q = \begin{pmatrix} p_{\text{src}_1}^\top \\ p_{\text{src}_2}^\top \\ \vdots \\ p_{\text{src}_n}^\top \end{pmatrix}$$

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

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

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

$$V = \begin{pmatrix} \text{MsgOut}_1 \\ \text{MsgOut}_2 \\ \vdots \\ \text{MsgOut}_{q_{\text{out}}} \end{pmatrix}$$

1065 By choosing a proper element-wise function φ , $\text{out}(\text{MachineIn}_R)_{i,1} = \text{output}_i$. □

1066 The theorem follows from stacking the elements from these three lemmas. Each lemma gives us a
1067 single layer of the final EMA-transformer T with embedding dimension $m = O(N^{5\varepsilon} \log N)$:

$$T = \text{out} \circ \text{route}_{R-1} \circ \cdots \circ \text{route}_1 \circ \text{init} \quad \square$$

1068 **Remark B.7** (General (γ, ε) -MPC). *The above simulation works the same for (γ, ε) -MPC by padding*
1069 *$\max(0, O(N^{1+\gamma-\varepsilon}) - N)$ empty chain-of-thought tokens in the input.*

1070 **Remark B.8** (Number of heads). *The standard transformer can simulate MPC using the same*
1071 *embedding dimension but only 1 attention head [51] [53]. Here the EMA needs $O(N^\varepsilon)$ heads, and we*
1072 *leave how to improve the number of heads for future work.*

1073 Since EM attention is a special case of ANN attention with $r = 0$ and $c \rightarrow \infty$, the simulation of
1074 MPC with ANNA-transformer naturally follows from Theorem B.2.

1075 **Corollary B.9** (ANNA simulates MPC). *For constant $0 < \varepsilon < 1$, any deterministic R -round*
1076 *MPC protocol π with N machines with $s = O(N^\varepsilon)$ words local memory, there exists an ANNA-*
1077 *transformer T with depth $L = R + 1$, number of heads $H = O(N^\varepsilon)$, and embedding dimension*
1078 *$m = O(N^{5\varepsilon} \log N)$, such that $T(\text{input}) = \pi(\text{input})$ for all $\text{input} \in \mathbb{Z}_{2^p}^N$.*

1079 Theorem B.2 only gives us an MPC simulation in the sublinear local memory regime when $s =$
1080 $O(N^{1/5-\delta})$ for any $\delta > 0$. However, a lot of MPC protocol algorithms require $s = \Omega(N^{1/2})$ local,
1081 such as MPC algorithm for 3-SUM [25] and algorithms for graphs [51]. The above simulation using
1082 EMA-transformer does not yield a sublinear embedding dimension. [51] further gives a simulation
1083 of MPC with sublinear local memory using transformer with sublinear embedding dimension by
1084 simulating one round of MPC protocol with $O(1)$ layers of transformer, instead of just one layer.
1085 Their improvement also applies here.

1086 **Theorem B.10** (EMA simulates MPC with improved embedding dimension). *For constant $0 < \varepsilon <$*
1087 *$\varepsilon' < 1$, any deterministic R -round MPC protocol π with N machines with $s = O(N^\varepsilon)$ words local*
1088 *memory, there exists an EMA-transformer of depth $L = O(R)$, number of heads $H = O(N^{\frac{\varepsilon'-\varepsilon}{4}})$*
1089 *and embedding dimension $m = O(N^{\varepsilon'})$ such that $T(\text{input}) = \pi(\text{input})$ for all $\text{input} \in \mathbb{Z}_{2^p}^N$.*

1090 *Proof.* The proof relies on simulating any MPC protocol by a restricted version of MPC protocol
1091 [51] which limits the number of machines each machine can send message to. Then, use a modified
1092 version of Theorem B.2 to simulate this restricted version of MPC.

1093 **Definition B.11** (Definition 3 of [51]). *For constants $\gamma, \varepsilon, \rho > 0$, a $(\gamma, \varepsilon, \rho)$ -MPC protocol is a (γ, ε) -*
1094 *MPC protocol with an additional constraint: in each round, each machine can only send/receive*
1095 *messages from $k = O(n^\rho)$ machines, while the total size of messages it can send and receive is still*
1096 *$s = O(N^\varepsilon)$. We refer to k as the communication capacity.*

1097 [51] gives a construction that simulates a R -round (γ, ε) -MPC protocol with $O(R)$ -round $(\gamma, \varepsilon, \rho)$ -
1098 MPC protocol. We restate their proposition here.

1099 **Lemma B.12** (Proposition 24 of [51]; $(\gamma, \varepsilon, \rho)$ -MPC simulates (γ, ε) -MPC). *For constants $\gamma, \varepsilon > 0$
1100 and $\rho \in (0, \varepsilon/2)$, if function f can be computed by an R -round (γ, ε) -MPC, it can also be computed
1101 by a $O(\frac{R(1+\gamma)^2}{\rho^2})$ -round $(\gamma, \varepsilon, \rho)$ -MPC protocol.*

1102 Therefore, we just need to simulate $(\gamma, \varepsilon, \rho)$ -MPC protocol using our EMA-transformer. The simu-
1103 lation follows the same recipe as Theorem B.2 where we have the initialization, message passing
1104 and final output phase. Since the initialization and output of $(\gamma, \varepsilon, \rho)$ -MPC follow the same rule as
1105 (γ, ε) -MPC, we only need to modify the message passing part of the simulation which corresponds to
1106 the routing Lemma B.3.

1107 **Lemma B.13** (EMA simulates $(\gamma, \varepsilon, \rho)$ -MPC). *For constant $0 < \rho < \varepsilon < 1$, any deterministic R -
1108 round MPC protocol π with N machines with $s = O(N^\varepsilon)$ words local memory and communication
1109 capacity $k = O(N^\rho)$, there exists an EMA-transformer of depth $L = R + 1$, number of heads
1110 $H = O(N^\rho)$ and embedding dimension $m = O(N^{\varepsilon+4\rho} \log N)$ such that $T(\text{input}) = \pi(\text{input})$
1111 for all $\text{input} \in \mathbb{Z}_{2^p}^N$.*

1112 *Proof.* For the initialization and final output part, we just use the same `init` and `out` constructed in
1113 Lemma B.5 and Lemma B.6. In the routing part (Lemma B.3), because $|\text{sent}^i| \leq k, |\text{rcvd}^i| \leq k$,
1114 we only need k heads and in each head of EMA, there are at most k keys matching each query and
1115 thus at most k values get averaged for a single query. Therefore, we can apply Lemma B.4 with
1116 $\alpha = k$ and $\Delta = s$, leading to an embedding dimension $m = O(N^{\varepsilon+4\rho} \log N)$. This gives us a new
1117 `route'_r` with reduced number of heads and embedding dimension for each round r .

1118 Likewise, we stack the 3 building blocks of one-layer EMA-transformer and have an $(R + 1)$ -layer
1119 EMA transformer

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

1120 and this finishes the construction for the lemma. \square

1121 Let $\rho = \min(\varepsilon/2, (\varepsilon' - \varepsilon)/4)$. In this setting, $\gamma = \varepsilon$. By Lemma B.12, we can simulate the R -round
1122 (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
1123 simulate this R' -round $(\gamma, \varepsilon, \rho)$ -MPC by an $R' + 1$ -layer EMA transformer with $O(N^\rho)$ heads and
1124 embedding dimension $O(N^{\varepsilon+4\rho} \log N) = O(N^{\varepsilon'})$. \square

1125 Again, the improved simulation result of ANNA-transformer follows from Theorem B.10.

1126 **Corollary B.14** (ANNA simulates MPC with improved embedding dimension). *For constant $0 < \varepsilon <$
1127 $\varepsilon' < 1$, any deterministic R -round MPC protocol π with N machines with $s = O(N^\varepsilon)$ words local
1128 memory, there exists an ANNA-transformer of depth $L = O(R)$, number of heads $H = O(N^{\frac{\varepsilon' - \varepsilon}{4}})$
1129 and embedding dimension $m = O(N^{\varepsilon'})$ such that $T(\text{input}) = \pi(\text{input})$ for all $\text{input} \in \mathbb{Z}_{2^p}^N$.*

1130 C MPC can Simulate ANNA-transformer

1131 As a warm-up, we first simulate EMA-transformers using MPC, and then generalize it to the
1132 simulation of ANNA-transformers.

1133 **Theorem C.1** (MPC simulates EMA). *Fix constants $0 < \varepsilon < \varepsilon' < 1$. For any L -layer EMA-
1134 transformer T with $mH = O(N^\varepsilon)$, there exists a $O(\frac{L}{\varepsilon' - \varepsilon})$ -round MPC protocol π with local
1135 memory $s = O(N^{\varepsilon'})$ and $P = O(N^{1+\varepsilon-\varepsilon'})$ machines such that $\pi(\text{input}) = T(\text{input})$ for all
1136 $\text{input} \in \mathbb{Z}_{2^p}^N$.*

1137 *Proof.* We first show how to use MPC to simulate one layer of EMA-transformer. In the high level,
1138 for each token x_i , we have a token Machine i which is responsible for computing the key, query and
1139 value embedding for x_i and other element-wise computation on x_i . The main bulk of the proof is to
1140 search for the exact matching keys for each query and send the averaged values associated with the
1141 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 $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_N)$ and a key ranking permutation of $[N]$ by $\sigma' = (\sigma'_1, \sigma'_2, \dots, \sigma'_N)$ such that

$$q_{\sigma_1} < q_{\sigma_2} < \dots < q_{\sigma_N} \\ \text{and } k_{\sigma'_1} < k_{\sigma'_2} < \dots < k_{\sigma'_N}$$

For the “meta-info” machine, we use a uniform hash function $h : \mathcal{X} \rightarrow [N]$ 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 i th 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 $\{q_{\sigma_{(i-1) \cdot n_q + 1}}, \dots, q_{\sigma_{i \cdot n_q}}\}$.
- 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 $\{(k_{\sigma'_{(i-1) \cdot n_q + 1}}, v_{\sigma'_{(i-1) \cdot n_q + 1}}), \dots, (k_{\sigma'_{i \cdot n_q}}, v_{\sigma'_{i \cdot n_q}})\}$.
- 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 $\text{start} = \arg \min_j \{q \in \text{Machine}(j, Q)\}$, and the end machine index, i.e. (end, Q) where $\text{end} = \arg \max_j \{q \in \text{Machine}(j, Q)\}$.
- 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 $k \in h_k^i$, this machine stores the start machine index, i.e. (start, KV) where $\text{start} = \arg \min_j \{k \in \text{Machine}(j, KV)\}$, and the end machine index, i.e. (end, Q) where $\text{end} = \arg \max_j \{k \in \text{Machine}(j, KV)\}$.
- 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 [53]), 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 $(\lceil 2mi/s \rceil, KV)$.

1184 Then, sorting query data structure machines $((i, Q)$ for all $i \in \lceil mN/s \rceil$) sorts the queries. Sorting in
 1185 MPC has been well studied, and this can be done in constant number of rounds [22].

1186 **Lemma C.2** (MPC Sorting). *There exists an MPC protocol with local memory $s = O(N^{\varepsilon'})$ that can*
 1187 *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.*

1188 After sorting, for each $i \in [N]$, we need to send the location information of q_i , ie which data structure
 1189 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
 1190 structure to aggregate the information and each query data structure machine is a leaf node of this
 1191 tree. Recall that each machine (i, Q) stores the queries $S = \{q_{\sigma_{(i-1) \cdot n_q + 1}}, \dots, q_{\sigma_{i \cdot n_q}}\}$. If S contains
 1192 the start and end of a particular query vector q_l , then (i, Q) sends a message $(q_l, (i, Q))$ to machine
 1193 $(h(q_l), h_q)$. Machine (i, Q) also sends the first and last query to its parent machine in the tree, i.e.
 1194 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
 1195 all the messages from the leaf node, it then does the same as its child: if it contains the start and end
 1196 of a certain query q , it sends to the location information (the first and last machine that store it) of the
 1197 query to its corresponding “meta-info” machine $(h(q), h_q)$, and it sends the first and last query and
 1198 their location information to its parent machine. This is done recursively, and since there are $\lceil mN/s \rceil$
 1199 query data structure machines in total, the depth of this $\frac{s}{m}$ -ary tree is $O(\log_{s/m} mN/s) = O(\frac{1}{\varepsilon' - \varepsilon})$,
 1200 which means $O(\frac{1}{\varepsilon' - \varepsilon})$ rounds and $O(mN/s)$ machines suffice.

1201 We do the same for (k, v) pairs. The sorting data structure machines (i, KV) sort the (k, v) pairs
 1202 based on the order of k . As before, we build a $\frac{s}{2m}$ -ary tree to send the location information to the
 1203 “meta-info” machine of each key. The different part from query is that we combine the values that
 1204 have the same key. For each machine in the $\frac{s}{2m}$ -ary tree, it computes the averaged value associated
 1205 with each key it contains and sends the averaged value to the corresponding “meta-info” machine. In
 1206 particular, for each k_i , the “meta-info” machine for k_i , $(h(k_i), h_k)$, contains the information (k_i, \bar{v})
 1207 where \bar{v} is the average of v_j 's such that $k_j = k_i$.

1208 Next, the “meta-info” machines of query and key need to exchange information to retrieve the
 1209 corresponding value for each query. Each (i, h_k) sends the (k, \bar{v}) pairs it has to the machine (i, h_q) .
 1210 Then, each (i, h_q) machine matches the q and k , and sends the associated value \bar{v} to the q . Note
 1211 that this step can be done by back propagating the $\frac{s}{m}$ -ary tree constructed for sending the location
 1212 information of q to $(h(q), h_q)$. In other words, we can just reverse the message sending direction in
 1213 this tree. Therefore, each query in the query data structure machine receives the value it retrieves and
 1214 from each query data structure machine (i, Q) , we can send the retrieved value for each query to its
 1215 corresponding token machine, which is the inverse of the second round.

1216 To summarize, the total rounds needed is $O(\frac{1}{\varepsilon' - \varepsilon})$ and the number of machines needed is $O(mN/s) =$
 1217 $O(N^{1+\varepsilon - \varepsilon'})$. To make this work for H heads, we can create H copies of this and each copy runs
 1218 in parallel. Since $mH = O(N^\varepsilon)$, the bounds for number of rounds and machines still hold. By
 1219 creating this MPC simulation for each of the L -layers, we stack them in the order of layers yielding
 1220 the complete simulation for L -layer EMA-transformer. \square

1221 Next, we generalize the above algorithm and proceed to simulate the ANN attention that can be
 1222 computed by Algorithm 1. Since Algorithm 1 is a randomized algorithm, we assume that the MPC
 1223 protocol shares all the random seeds needed for all the layers of ANNA-transformer.

1224 **Theorem C.3** (MPC simulates ANNA). *Fix constants $0 < \varepsilon < \varepsilon' < 1$. For any L -layer ANNA-*
 1225 *transformer T (as implemented by Algorithm 1) with $mH = O(N^\varepsilon)$, there exists a $O(L/(\varepsilon' - \varepsilon))$ -*
 1226 *round MPC protocol π with local memory $s = O(N^{\varepsilon'})$ and $P = O(N^{1+\varepsilon - \varepsilon' + 3/\varepsilon^2})$ machines such*
 1227 *that $\pi(\text{input}) = T(\text{input})$ for all $\text{input} \in \mathbb{Z}_{2^p}^N$.*

1228 *Proof.* The high level idea of simulating ANNA-transformer is very similar to simulating EMA. We
 1229 have the same kinds of machines as before. The biggest difference is that, instead of having one hash
 1230 table for queries and keys, we now have ℓ hash tables, one for each round of hashing, and we sort the
 1231 queries and keys based on the hash values of queries and keys. Again, we first outline different types
 1232 of machines we will use.

- 1233 • For $i \in [N]$, machine i is the *token machine* for x_i . This machine performs all the element-wise
1234 computation for token i . Specifically, it computes the query, key, value embeddings q_i, k_i, v_i and
1235 element-wise operations after the attention layer.
- 1236 • For $i \in [\lceil mN/s \rceil], t \in [\ell]$, machine (i, Q, h^t) is a data structure machine for sorted queries for
1237 the t -th hash table and the i -th chunk of the sorted list of queries, where the ordering of sorting is
1238 based on $g_t(q)$ from Algorithm 1.
- 1239 • For $i \in \lceil 2mN/s \rceil$, Machine (i, KV, h^t) is a data structure machine for sorted list of key and value
1240 pairs for the t -th hash table and the i -th chunk of the sorted list of key and value pairs, where the
1241 ordering of sorting is based on $g_t(k)$.
- 1242 • 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
1243 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
1244 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
1245 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
1246 end machine index, i.e. (end, Q, h^t) where $\text{end} = \arg \max_j \{q \in \text{Machine}(j, Q, h^t)\}$.
- 1247 • 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
1248 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
1249 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
1250 start machine index, i.e. (start, KV, h^t) where $\text{start} = \arg \min_j \{k \in \text{Machine}(j, KV, h^t)\}$, and
1251 the end machine index, i.e. (end, Q, h^t) where $\text{end} = \arg \max_j \{k \in \text{Machine}(j, KV, h^t)\}$.
- 1252 • The auxiliary machines needed for message propagation.

1253 Like before, we still use each token machine to compute the embeddings $q_i, k_i, v_i \in \mathbb{R}^m$. Then,
1254 each token machine need to send (q_i, i) and (k_i, v_i, i) to the data structure machines, machine
1255 $(\lceil mi/s \rceil, Q, h^t)$ 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
1256 tree to propagate the queries and keys to the corresponding data structure machines. This takes
1257 $O(\frac{1}{\varepsilon'^{-\varepsilon}})$ rounds and $O(N^{1+3\rho+\varepsilon-\varepsilon'})$ machines.

1258 Then, for each query hash table $t \in [\ell]$, the data structure machines sort the queries based on the hash
1259 value of the queries. Same as Theorem C.1, we use the $\frac{s}{m}$ -ary tree to send the location information
1260 of each hash bucket to its corresponding “meta-info” machine. For each key, value pair hash table
1261 $t \in [\ell]$, the data structure machines sort the key, value pairs based on the hash value of the keys. After
1262 that, use the $\frac{s}{2m}$ -ary tree to propagate the information to the corresponding “meta-info” machine.
1263 The difference from the EMA simulation is that each machine in this $\frac{s}{2m}$ -ary tree maintains the sum
1264 of values whose key has the same hash values instead of the averaged value, and also maintains a
1265 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
1266 machines.

1267 Next, the key “meta-info” machine send the sum of values and count to the corresponding query
1268 “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
1269 “meta-info” machine then follows the $\frac{s}{m}$ -ary tree, broadcasting the sum of values and counts to the
1270 queries in the hash table. finally, each query in the hash table needs to propagate the information
1271 back to its original token machine. Since each token machine will receive message from $\ell = N^{3\rho}$
1272 machines, we again reverse the $\frac{s}{m}$ -ary tree that send the query to each data structure machine. During
1273 the aggregation, each machine in the $\frac{s}{m}$ -ary tree still maintains the sum of values and the sum of
1274 counts it receives. After receiving the sum of values and counts, each token machine i then calculates
1275 $\text{ANNA}(q_i) = \text{sum of the values divided by the counts}$.

1276 The above simulates one layer of ANNA-transformer in $O(\frac{1}{\varepsilon'^{-\varepsilon}})$ rounds and using $O(N^{1+3\rho+\varepsilon-\varepsilon'})$
1277 machines, where $\rho = 1/c^2$. Therefore, by stacking the simulation for L layers, this gives $O(\frac{L}{\varepsilon'^{-\varepsilon}})$
1278 rounds in total. To extend to H heads, we just need to instantiate the above simulation for H parallel
1279 copies and because $mH = O(\varepsilon)$, the total number of rounds and machines still remains the same. \square

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 $K'(X)^T V(X)$, recall that

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

We just need to compute the sum of N matrices of size $r \times m$. Each token machine i computes the matrix $k_i v_i^T$ 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 $M = K'(X)^T V(X) \in \mathbb{R}^{r \times m}$, in order to compute $Q(X)K'(X)^T V(X)$, we just need to propagate the matrix M to all the token machines. And each token machine i computes $q_i^T M$. 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. \square

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. \square

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:*

- 1318 1. Apply a hash function $h : \mathbb{R}^m \rightarrow U$ on $\{q_1, \dots, q_N\}$;
 1319 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 $B \leq O(1)$, and let
 1320 $h'(q_i)$ be the label of the chunk that q_i is in (the queries in each chunk can have different hash
 1321 values);
 1322 3. For each q_i , only attend to k_j 's such that they are in the same chunk.

1323 The output embedding for q_i is therefore

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

1324 We define $f_\ell : [N] \rightarrow [N]^B$ as the function that specifies the set of keys each query should compute
 1325 inner product with in the ℓ -th layer. From the Reformer constraints, we have $\forall i \in [N]$:

- 1326 1. $f_\ell(i) = \{a_1, a_2, \dots, a_B\} \in [N]^B$ is a set (no repetition).
 1327 2. $i \in f_\ell(i)$.
 1328 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

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

1329 We first study a restricted version of Reformer that fix the communication pattern beforehand i.e. f_ℓ
 1330 is input-independent for all $\ell \in [L]$, and show that it can not compute the sum of all the input tokens.

1331 **Definition E.2 (SUM).** Given input $X = (x_1, x_2, \dots, x_N)$, $x_i \in [M]$, and $M = N^{O(1)}$, the
 1332 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 ,
 1333 $T(X)_N = \text{SUM}(X)$.

1334 Here $T(X)_N$ is the N -th output of T given the input X . One can choose any position to be the final
 1335 output position, and here WLOG we choose the last token to follow the autoregressive generation
 1336 model convention.

1337 **Proposition E.3.** Fix $L = O(1)$ and $\{f_\ell\}_{\ell=1}^L$. Any Reformer T with L layers and each layer the
 1338 attention pattern is specified by $\{f_\ell\}_{\ell=1}^L$ can not compute SUM(X): there exists an X , $|T(X)_N -$
 1339 $\text{SUM}(X)| \geq \epsilon$, for any $0 < \epsilon < M/2$.

1340 *Proof.* We denote each layer's element-wise computation by $\{\phi_\ell\}_{\ell=1}^L$. Let $T^\ell(X)_i$ denote the i -th
 1341 output of T after ℓ layers of computation. We prove this proposition by induction.

1342 Inductive hypothesis: $T^\ell(X)_i$ is a function of at most B^ℓ different $x_i \in X$.

1343 Base case: $\ell = 1$

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

1344 which is a function of at most B x_i 's in X .

1345 Inductive step: consider

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

1346 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
1347 $B \cdot B^\ell = B^{\ell+1}$ variables from X .

1348 Therefore, if $T^L(X)_i$ is a function of all $\{x_1, \dots, x_N\}$, we need $B^L \geq N$ and thus $L = \Omega(\log_B N)$.
1349 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
1350 $T^L(X)_N$ is a function of $\{x_1, \dots, x_{B^L}\}$. Then, x_{B^L+1} can be any number in $[M]$ that makes
1351 $T^L(X)_N$ far from $\text{SUM}(X)$. \square

1352 Therefore, if Reformer has any power, it must come from the sorting part, because the sorting
1353 algorithm have access to the information of all the token inputs.

1354 Although constant-layer Reformer can not compute SUM, one can easily show that one layer of
1355 ANNA-transformer can compute SUM by setting $v_i = Nx_i$ and $k_1 = k_2 = \dots = k_N = q_N$ for all
1356 $i \in [N]$, thereby retrieving all the v_i 's and averaging them.

1357 F ANNA-transformer Solves k -hop and Match2

1358 F.1 ANNA/EMA-transformer Solves Match2

1359 **Theorem F.1.** For any $N, M = N^{O(1)}$, there exists an EMA-transformer T with one layer, one
1360 attention head, and embedding dimension 1 such that $T(X) = \text{Match2}(X)$ for all $X \in [M]^N$.

1361 *Proof.* Given input $X \in [0, M]^{N \times 1}$. Let $Q(X) = \phi(X)Q, K(X) = \phi(X)K, V(X) = \phi(X)V$,
1362 where Q, K, V are matrices in $\mathbb{R}^{2 \times 1}$. Define ϕ by $\phi(x) = (x, 1)$ and

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

1363 such that

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

1364 As a result, for each $1 \leq i \leq N$, if there exists $1 \leq j \leq N$ such that $x_i + x_j = M$, then

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

1365 Otherwise, $((\phi(X)Q)(\phi(X)K)^\top)[i, j] = 0$ for all $1 \leq j \leq N$. Finally, we can calculate that if
1366 $|\{j \in [N] : x_i + x_j = M\}| \neq 0$, then

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

1367 and if $|\{j \in [N] : x_i + x_j = M\}| = 0$, then

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

1368 That gives the same result for ANNA-transformers.

1369 **Corollary F.2.** For any $N, M = N^{O(1)}$, there exists an ANNA-transformer T with one layer, one
1370 attention head, and embedding dimension 1 such that $T(X) = \text{Match2}(X)$ for all $X \in [M]^N$.

1371 F.2 ANN transformer Solves k -hop

1372 We first show that ANNA transformers can solve induction head (1-hop).

1373 **Lemma F.3.** Fix constants $0 < \varepsilon < \varepsilon' < 1$, and $|\Sigma| \leq N$. There exists an ANNA-transformer
 1374 T with $L = O(\frac{1}{\varepsilon \cdot (\varepsilon' - \varepsilon)^2})$ layers, $H = O(N^{(\varepsilon' - \varepsilon)/4})$ heads per layer, and embedding dimension
 1375 $m = O(N^{\varepsilon'})$ 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
 1376 all $w \in \Sigma^N$.

1377 *Proof.* We prove this lemma by designing a constant-round MPC algorithm with local memory
 1378 $s = O(N^\varepsilon)$ and N/s machines to solve 1-hop. Since $|\Sigma| \leq N$, each token can be embedded with
 1379 $O(\log N)$ bits ($O(1)$ words). Denote the input $w^N = (x_1, x_2, \dots, x_N)$. The MPC algorithm works
 1380 as following:

- 1381 1. For each x_i , retrieve the next token x_{i+1} and each token on the machine is stored as the embedding
 1382 of $(x_i, i, x_{i+1}, i+1)$.
- 1383 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
 1384 $(x_j, j, x_{j+1}, j+1)$, 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
 1385 $x_i = x_j$, 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
 1386 comparator $<$.
- 1387 3. Each token $(x_i, i, x_{i+1}, i+1)$ in the sorted list retrieves the token before it in the sorted list,
 1388 denoted by $(x_j, j, x_{j+1}, j+1)$. Update the embedding of token: if $x_j = x_i$, the embedding of
 1389 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
 1390 of the token x_i becomes $(i, \perp, 0)$.
- 1391 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 .

1392 For step 1, each machine only needs to send message to its neighbor machine: machine i sends
 1393 message to machine $i-1$, and this only takes 1 round. In step 2, each tuple is only $O(\log N)$ bits,
 1394 so by Lemma C.2 the sorting takes $O(\frac{1}{\varepsilon})$ rounds. In step 3, again each machine only needs to send
 1395 message to its neighbor machine: machine i sends message to machine $i+1$, and this only takes
 1396 1 round. In step 4, each machine for the sorted list sends at most s tuples stored in it to the correct
 1397 output machine which takes 1 round. Thus, the MPC algorithm has $O(\frac{1}{\varepsilon})$ rounds in total.

1398 Then, we convert this MPC algorithm to an ANNA-transformer. By Corollary B.14, this gives us an
 1399 ANNA-transformer with number of heads $H = O(N^{(\varepsilon' - \varepsilon)/4})$, embedding dimension $m = O(N^{\varepsilon'})$
 1400 and number of layers $L = O(\frac{1}{\varepsilon \cdot (\varepsilon' - \varepsilon)^2})$. \square

1401 Now we show that ANNA-transformers can solve k -hop with $O(\log k)$ layers.

1402 **Theorem F.4.** Fix constants $0 < \varepsilon < \varepsilon' < 1$, $|\Sigma| \leq N$ and any $k \in \mathbb{N}$. There exists an ANNA-
 1403 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
 1404 embedding dimension $m = O(N^{\varepsilon'})$ such that $T(w)_i = w_{\sigma^k(w,i)}$, $\forall i \in [N]$, for all $w \in \Sigma^N$.

1405 *Proof.* We prove this theorem by constructing an $O(\log k)$ -rounds MPC with $s = O(N^\varepsilon)$ local
 1406 memory and $O(N/s)$ machines. We show that this MPC algorithm can compute k -hop by induction.
 1407 Let $k = \sum_{j=0}^{\lceil \log k \rceil} k_j 2^j$ 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

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

1408 Base case: $\ell = 0, k = 1$ is implied by Lemma F.3. After step 3, we have $(i, w_{\sigma(w,i)}, \sigma(w,i))$.
 1409 Now consider $k = \ell + 1$. For each i , the machine (Machine $\lceil i/s \rceil$) that contains

1410 $(i, w_{\sigma^{2^\ell}(w,i)}, \sigma^{2^\ell}(w,i), w_{\sigma^{k:\ell}(w,i)}, \sigma^{k:\ell}(w,i))$ sends the message $(i, \sigma^{2^\ell}(w,i))$ to machine that con-
 1411 tains $\sigma^{2^\ell}(w,i)$ 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
 1412 send the following tuple to machine $\lceil i/s \rceil$:

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

1413 Since each machine has at most s tuples and the function $\sigma(w,i)$ is one-to-one except for \perp , the
 1414 number of messages each machine sends and receive is bounded by s . After machine $\lceil i/s \rceil$ receiving
 1415 the above message, it update the tuple for the token i :

- 1416 1. if $k_\ell = 0$, token i is updated as: $(i, t_1, t_2, w_{\sigma^{k:\ell}(w,i)}, \sigma^{k:\ell}(w,i))$
 1417 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:

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

1418 The above inductive step only takes 2 rounds of MPC. Therefore, the total round is $O(\frac{1}{\varepsilon}) + 2(\ell + 1)$.
 1419 When $\ell = \lfloor \log k \rfloor + 1$, this algorithm compute the output for k -hop.

1420 Again, we can convert this MPC algorithm to an ANNA-transformer. By Corollary [B.14](#), this
 1421 gives us an ANNA-transformer with number of heads $H = O(N^{(\varepsilon' - \varepsilon)/4})$, embedding dimension
 1422 $m = O(N^{\varepsilon'})$ and number of layers $L = O(\frac{1}{\varepsilon \cdot (\varepsilon' - \varepsilon)^2} + \frac{\log k}{(\varepsilon' - \varepsilon)^2})$. \square

1423 G Experimental details

1424 Here are the details of the experimental setup. All the experiments are launched on 2 GPUs: NVIDIA
 1425 Titan RTX and NVIDIA Titan Xp.

1426 We train a modified version of the attention matrix and then distill from the trained model us-
 1427 ing ANNA implemented by the angular distance LSH family from [\[6\]](#). Our softmax atten-
 1428 tion normalizes all the queries and keys in $Q(X)$ and $K(X)$ to have unit norm, and computes
 1429 $\text{softmax}(\beta \cdot Q(X)K(X)^\top)V(X)$ with a hyperparameter $\beta > 0$.

1430 G.1 Match2 experiments

1431 **Dataset generation.** Inspired by the way [\[36\]](#) generating data for Match3 task, a triple-wise version
 1432 of Match2, we generate the data for Match2 using the same algorithm but change to pair-wise relation
 1433 when computing the label. Each sample is a tuple (X, Y) , where $X = (x_1, x_2, \dots, x_N)$, and each x_i
 1434 is an integer sampled from $\{1, 2, \dots, 36\}$; $Y = (y_1, y_2, \dots, y_N)$, and each $y_i = \mathbb{1}\{\exists j \cdot x_i + x_j =$
 1435 $0 \bmod 37\}$. The sequence length N is set to 32. When sampling the data, we ensure that each batch
 1436 is balanced by having the distribution of one's in Y the same: each batch has 4 bins and each bin
 1437 corresponds to each percentage $[0, 25\%)$, $[25\%, 50\%)$, $[50\%, 75\%)$, $[75\%, 100\%)$ of one's in Y ; each
 1438 bin size is $1/4$ of the batch size. See Algorithm [2](#) for details.

1439 **Training details.** We trained 3 models with $\beta \in \{0.1, 1, 10\}$ respectively, with Adam optimizer on
 1440 cross-entropy loss and learning rate 0.01. Each model has one layer, one attention head, embedding
 1441 dimension $m = 64$ and an MLP with width $4m$ and GeLU activation. The dataset size, batch size,
 1442 training steps are 10000, 32, 20000 respectively.

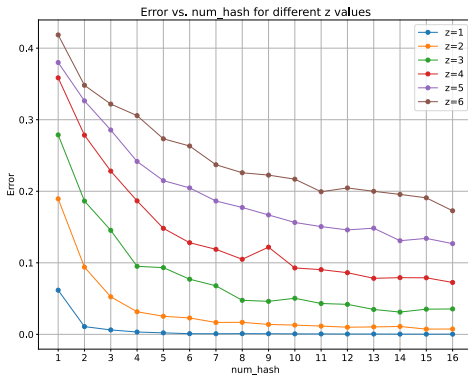
1443 We apply ANNA with number of hash tables $\ell \in \{1, 2, \dots, 16\}$ and number of hash functions for
 1444 each table $z \in \{1, 2, \dots, 6\}$ on all the 3 trained models, and $\beta = 0.1$ has the best performance (error
 1445 can be 0). Since the implementation of ANNA is randomized, for each combination of (ℓ, z) , we run
 1446 10 times and report the averaged error over the 10 runs. See Figure [1a](#) for plotted performance when
 1447 $\beta = 0.1$. In this setting, $\ell \geq 8, z = 1$ can achieve 0 error on the test set with 256 test samples.

Algorithm 2 Match2 Dataset Generation

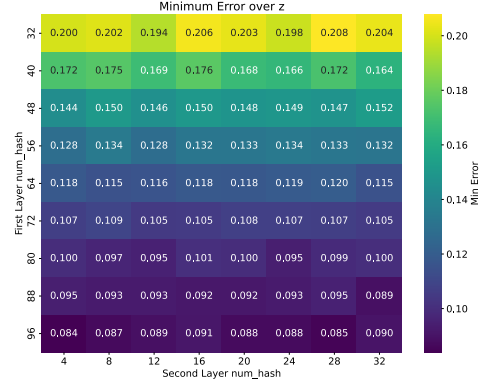
Input: $N = 32, M = 37$, dataset size D

Output: Dataset of (X, Y) pairs

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



(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 $z \in \{1, 2, 3, 4\}$.

1448 G.2 Induction heads experiments

1449 **Dataset generation.** We use the data generation algorithm from [53]. Each sample (X, Y) is of
 1450 the form $X = (k, X')$ for $k \in \{0, 1\}$ for training samples and $k = 1$ for test samples, $X' \in \Sigma^{N-1}$,
 1451 $Y = (0, Y')$, where Y' is the k -hop label for X' . Here the sequence length $N = 100$ and alphabet
 1452 size $|\Sigma| = 4$.

1453 **Training details.** We trained 3 models with $\beta \in \{0.1, 1, 10\}$ respectively, with Adam optimizer
 1454 on cross-entropy loss and learning rate 0.01. Each model has 2 layers, each layer with one attention
 1455 head, embedding dimension $m = 128$ and an MLP with width $4m$ and GeLU activation. We use
 1456 online training: at each training step, sample fresh new data to train. The batch size and training steps
 1457 are 32, 400000 respectively.

1458 We apply ANNA on all the 3 trained models. For the first layer, the number of hash tables ℓ is chosen
 1459 from $\{32, 40, 48, \dots, 96\}$ and z is chosen from $\{1, 2, 3, 4\}$. For the second layer, the number of hash
 1460 tables ℓ is chosen from $\{4, 8, 12, \dots, 32\}$ and z is chosen from $\{1, 2, 3, 4\}$. When evaluating the test
 1461 error, we compute the error on all the tokens. Note that this is different from [53], where they only

1462 compute the error on the tokens whose induction head exists to avoid overestimating the performance
1463 when k is large and has a large fraction of null outputs. In our setting, $k = 1$ which doesn't have
1464 many null outputs, and it is important for the model to learn when to output the null token.

1465 We found $\beta = 1$ has the best performance, so we report it in Figure 1b. Again, the errors are averaged
1466 over 10 runs for each combinations and taken the minimum over z 's. One can see that 32 hash tables
1467 in the first layer and 4 hash tables in the second layer already gives highly non-trivial performance:
1468 the error rate is 0.2 over 100 samples and each sample has 100 token predictions, while random guess
1469 would give 0.75 error rate. With more hash tables in the first layer, the error rate can go below 0.1.