H-Transformer-1D: Fast One-Dimensional Hierarchical Attention for Sequences

Zhenhai Zhu
Google Research
zhenhai@google.com

Radu Soricut
Google Research
rsoricut@google.com

arXiv:2107.11906v1 [cs.LG] 25 Jul 2021

Abstract

We describe an efficient hierarchical method to compute attention in the Transformer architecture. The proposed attention mechanism exploits a matrix structure similar to the Hierarchical Matrix (H-Matrix) developed by the numerical analysis community, and has linear run time and memory complexity. We perform extensive experiments to show that the inductive bias embodied by our hierarchical attention is effective in capturing the hierarchical structure in the sequences typical for natural language and vision tasks. Our method is superior to alternative sub-quadratic proposals by over +6 points on average on the Long Range Arena benchmark. It also sets a new SOTA test perplexity on One-Billion Word dataset with 5x fewer model parameters than that of the previous-best Transformer-based models.

1 Introduction

Linearly combining information using content-based weights, a method generically known as attention, is a key building block in many deep neural networks such as recurrent neural networks (RNN) (Luong et al., 2015), convolutional neural networks (CNN) (Bello et al., 2019) and graph convolutional networks (GCN) (Velickovic et al., 2018). One particular type of such attention, called multi-head scaled dot-product attention, is one of the main components of the Transformer architecture proposed by Vaswani et al. (2017), which has been shown to push the state-of-the-art (SOTA) performance for various understanding and generation tasks. These include standard natural language processing (NLP) tasks such as machine translation, document classification, entailment, summarization and question answering (Zaheer et al., 2020; Dai et al., 2019; Baevski and Auli, 2019), as well as music generation (Huang et al., 2018), image generation (Parmar et al., 2018; Chen et al., 2020) and genomics (Zaheer et al., 2020; Choromanski et al., 2020). The Transformer is also the backbone architecture for models such as BERT (Devlin et al., 2019) (and its numerous relatives) and GPT3 (Brown et al., 2020), which have delivered impressive performance across many NLP tasks. However, the standard attention mechanism of the Transformer has a run time and memory usage that scales quadratically with sequence length. Therefore, this quadratic complexity has become a critical bottleneck in processing long sequences (over 1,000 tokens), and has since motivated many new attention algorithms, see (Tay et al., 2020d) for a survey of such work.

In this paper, we draw inspiration from two branches in numerical analysis: Hierarchical Matrix (H-Matrix) (Hackbusch, 1999, 2000) and Multigrid method (Briggs et al., 2000). We propose a hierarchical attention that has linear complexity in run time and memory, and only utilizes dense linear algebra operations optimized for GPUs or TPUs.

We hypothesize that the inductive bias embodied by the proposed hierarchical structure for the attention matrix is effective in capturing the hierarchical structure in the sequences typically seen in many natural language processing and computer vision tasks. The main benchmark we use in this paper is the Long Range Arena (LRA) benchmark (Tay et al., 2020c), which has been specifically designed to evaluate and compare various sub-quadratic attention algorithms. Our new hierarchical attention mechanism achieves best average performance to-date on the LRA benchmark by more than 6 points over the previous-best BigBird algorithm (Zaheer et al., 2020), while pushing SOTA performance higher

in 4 of the 5 successful tasks. Furthermore, using this new attention, a Transformer-based language model trained on the One-Billion Word dataset (Chelba et al., 2014) sets a new SOTA performance record by reducing the test perplexity by 1.55 points comparing to the previous-best Transformer-XL (Dai et al., 2019) with 5x more parameters. Overall, these empirical results both validate the soundness of our approximation method for computing attention weights, as well as the the appropriateness of the inductive bias present in the proposed hierarchical attention.

2 Related Works

It is well established in the NLP literature that the embeddings of nearby tokens tend to be more similar than the distant ones (Manning and Schütze, 1999). This leads to the intuition that token similarity and hence the attention should decrease with the sequence distance between a query token and a key token[1]. This motivates the sliding-window local attention (Parmar et al., 2018; Ramachandran et al., 2019; Qiu et al., 2019) which amounts to truncating off-diagonal entries in the attention matrix beyond a user-specified sequence distance. A second approach is to keep $O(1)$ number of nonzeros per row in the attention matrix. The nonzero entry selection is either content-based (Kitaev et al., 2020; Roy et al., 2020; Tay et al., 2020b; Zhou et al., 2020), handcrafted (Beltagy et al., 2020; Brown et al., 2020; Child et al., 2019; Ho et al., 2019) or simply random (Zaheer et al., 2020). It is also well known in the NLP literature that long-range contextual information is necessary for many NLP tasks (Khandelwal et al., 2018; Liu and Lapata, 2019). So a set of global tokens are also considered. This adds $O(1)$ number of dense rows and columns to the attention matrix (Zaheer et al., 2020; Ainslie et al., 2020; Beltagy et al., 2020). A third approach is to approximate the attention matrix with a low-rank factored form (Choromanski et al., 2020; Wang et al., 2020; Tay et al., 2020a).

The first two approaches are based on the premise that one needs to explicitly zero out entries in the attention matrix in order to reduce the quadratic complexity. Decades of research by the scientific computing and numerical analysis community has resulted in more sophisticated algorithms to sparsify matrices. A small set of samples of these algorithms and their engineering applications include Fast Multipole Method (Greengard and Rokhlin, 1987; Greengard, 1994; Nabors et al., 1994; Shi et al., 1998), Pre-corrected FFT (Phillips and White, 1997; Zhu et al., 2005), Hierarchical Singular Value Decomposition (SVD) (Kapur and Long, 1997) and Hierarchical Matrix (H-Matrix) (Hackbusch, 1999, 2000; Zhu and White, 2005). These are generally called Multilevel Methods (Brandt and Lubrecht, 1990). The hierarchical attention proposed in this paper is inspired by these Multilevel Methods in general and the H-Matrix in particular. The hierarchical matrix structure allows a linear complexity in both constructing and applying the attention matrix.

3 Definition and Notation

Given matrices Q, K and V, with rows representing sequences of token embedding or feature vectors for query, key and value respectively, the output weighted by the scaled dot-product attention in the Transformer (Vaswani et al., 2017) is defined as

$$Z = \mathbf{softmax}(\frac{QK^T}{\sqrt{d}})V \tag{1}$$

where $Z, Q, K, V \in R^{L\times d}$, L is the length of the sequences, and d is the embedding or feature size. In a more compact matrix form, Eq. (1) can be written as

$$Z = D^{-1}AV \tag{2}$$

where

$$A = e^S \tag{3}$$

$$S_{i,j} = \frac{Q_i K_j^T}{\sqrt{d}} \tag{4}$$

$$D = \mathbf{diag}\{A \cdot \mathbf{1}_L\} \tag{5}$$

$$1_L = [1, 1, ..., 1]^T. \tag{6}$$

Here, $A, S \in R^{L\times L}$, $\mathbf{1}_L \in R^L$ is a vector with all ones, and $S_{i,j}$ represents the unnormalized cosine similarity between query embedding Q_i (the i-th row in Q) and key embedding K_j (the j-th row in K).

For the sake of clarity, we focus on the single-head attention in the exposition of the proposed

[1]Eq. (11) and (12) offer a simple illustration of this intuition.

algorithm. Extension to the multi-head case is straightforward since each attention head is computed independently (Vaswani et al., 2017).

Computing the similarity matrix S in Eq. (4) and the attention matrix A in Eq. (3) takes $O(L^2d)$ time and $O(L^2)$ memory. Similarly, computing AV in Eq. (2) takes $O(L^2d)$ time, and computing $A \cdot \mathbf{1}_L$ in Eq. (5) takes $O(L^2)$ time. The $O(L^2d)$ and $O(L^2)$ complexities are the bottlenecks for applying the attention mechanism over very long sequences.

4 Introduction on H-Matrix and Multigrid Method

4.1 H-Matrix

The singular-value decomposition of the attention matrix A in Eq. (3) is

$$A = U\Sigma V^T \tag{7}$$

where $\Sigma = \mathbf{diag}\{\sigma_1, \sigma_2, ..., \sigma_L\}$ and σ_i is the i-th singular value. The numerical rank of matrix A is r if $\sum_{i=r+1}^{L} \sigma_i < \epsilon$ for a given tolerance ϵ (Trefethen and Bau, 1997). The standard rank-r approximation to matrix A is

$$A \approx \hat{U}\hat{\Sigma}\hat{V}^T = \hat{U}\tilde{V}^T \tag{8}$$

where $\hat{\Sigma} = \mathbf{diag}\{\sigma_1, \sigma_2, ..., \sigma_r\}$, $\hat{U}, \hat{V} \in R^{L\times r}$ have the first r columns of U and V, and $\tilde{V} = \hat{V}\hat{\Sigma}$. This is the low-rank approximation used in (Choromanski et al., 2020; Wang et al., 2020; Tay et al., 2020a). This approximation compresses L^2 entries in A to $2rL$ entries in $\hat{U}$ and $\tilde{V}^T$. So the compression rate is $\frac{L}{2r}$.

The H-Matrix generalizes this low-rank approximation by using matrix block hierarchy. Consider a two-level H-Matrix with 4×4 and 2×2 block partition at level-0 and level-1, respectively. Matrix A is partitioned as

$$A = \begin{bmatrix} \begin{array}{cc|cc} A_{11}^{(0)} & A_{12}^{(0)} & \multicolumn{2}{c}{\multirow{2}{*}{$A_{12}^{(1)}$}} \\ A_{21}^{(0)} & A_{22}^{(0)} & & \\ \hline \multicolumn{2}{c|}{\multirow{2}{*}{$A_{21}^{(1)}$}} & A_{33}^{(0)} & A_{34}^{(0)} \\ & & A_{43}^{(0)} & A_{44}^{(0)} \end{array} \end{bmatrix}. \tag{9}$$

The low-rank approximation in Eq. (8) is applied to the off-diagonal blocks at each level. For example,

$$A_{12}^{(l)} \approx \hat{U}_{12}^{(l)}(\tilde{V}_{12}^{(l)})^T \tag{10}$$

where $l = 0, 1$. To give a concrete example, suppose each entry in matrix A has the analytical form

$$A_{i,j} = e^{S_{i,j}} \tag{11}$$

$$S_{i,j} = 2e^{-(i-j)^2} - 1 \tag{12}$$

where $i, j = 0, 1, 2, ..., 15$ [2]. With the block hierarchy defined in Eq. (9), the size of the matrix block at level-1 and level-0 is 8×8 and 4×4, respectively. For tolerance $\epsilon = 10^{-3}$, one can verify that the numerical rank map of matrix A is

$$\begin{bmatrix} \begin{array}{cc|cc} 4 & 2 & \multicolumn{2}{c}{\multirow{2}{*}{2}} \\ 2 & 4 & & \\ \hline \multicolumn{2}{c|}{\multirow{2}{*}{2}} & 4 & 2 \\ & & 2 & 4 \end{array} \end{bmatrix} \tag{13}$$

where the number in each block is the numerical rank of the corresponding block in Eq. (9). Note that matrix A still has full numerical rank of 16 at a looser tolerance 10^{-1}. So the standard low-rank approximation is ineffective in this case. But even this simple two-level H-matrix already offers a compression rate of $\frac{4}{3}$ since storing an H-matrix with the rank map in Eq. (13) takes 192 entries [3]. In addition, one can verify that no entry $A_{i,j}$ in Eq. (11) is very small, since $S_{i,j} \in [-1, 1]$ in Eq. (12). Therefore, truncating off-diagonal entries of matrix A, as proposed in (Parmar et al., 2018), would produce a poor approximation. In practice, the number of levels is adapted to the underlining governing equations that result in matrix A and it can easily be over 10 (Kapur and Long, 1997; Hackbusch, 2000; Zhu and White, 2005). In turn, this can substantially increase the compression rate. In general, the computation complexity of the H-Matrix is either $O(L)$ or $O(L \log L)$, depending on the underlining physics (Hackbusch, 1999, 2000).

4.2 Elements of the Multigrid Method

Multigrid Method is a multi-level nested iterative method for solving large-scale sparse matrices resulting from discretized partial-differential equations (PDEs) (Briggs et al., 2000; Trottenberg et al., 2000). At its core are two

[2] Matrix A in Eq.(11) is a symmetric Toeplitz matrix (Golub and Loan, 1996) and hence only has 16 unique entries. But we ignore this fact and treat A as a general matrix here.

[3] Each one of four diagonal blocks at level-0 takes 16 entries. Each one of four off-diagonal blocks at level-0 takes 16 entries. Each one of two off-diagonal blocks at level-1 takes 32 entries.

simple but powerfully complementary ideas: relaxation and correction. Our proposed hierarchical attention only uses the correction scheme as a building block since there is no sparse matrix to relax on.

The correction scheme has two components: restriction or coarsening, and interpolation or prolongation. Consider a vector $\bar{v}^h$ of scalar values defined on a set of N grids with uniform interval h. The simplest coarsening is to take the average of the scalar values on each pair of grids, i.e.,

$$\bar{v}_j^{2h} = \frac{1}{2}(\bar{v}_{2j}^h + \bar{v}_{2j+1}^h) \tag{14}$$

where $j = 0, 1, 2, ...N/2 - 1$. The superscript in Eq. (14) indicates that the grid interval at these two levels is h and $2h$, respectively. The simplest interpolation is to duplicate the value on each coarse grid to values on a pair of fine grids, i.e.,

$$\bar{v}_{2j}^h = \bar{v}_j^{2h}, \quad \bar{v}_{2j+1}^h = \bar{v}_j^{2h} \tag{15}$$

where $j = 0, 1, 2, ...N/2 - 1$.

5 Intuition for Hierarchical Attention

The hierarchical low-rank structure like Eq. (13) turns out to be pervasive in many if not all physics phenomena. Much of the theoretical analysis by (Greengard and Rokhlin, 1987; Hackbusch, 1999) is concerned with quantifying such aspects. The key insight into these Multilevel Methods can be summarized as follows: *perform no approximation for near interactions, and apply progressively lower-precision approximation for progressively longer distance interactions*. The simple case shown in Eq. (9)-(13) is a good example. To satisfy the tolerance of 10^{-3}, we need full rank (no approximation) for the diagonal blocks (near interactions), higher precision approximation (rank-2 vs full-rank of 4) for the 4×4 off-diagonal blocks at level-0 (mid-distance) and lower precision approximation (rank-2 vs full-rank of 8) for the 8×8 off-diagonal blocks at level-1 (long-distance).

In this section, we present some intuition to answer two important questions: 1) Does the hierarchical low-rank structure hold for the attention matrix A in Eq. (3)? 2) What is the algorithm to efficiently compute the hierarchical low-rank structure? We only give an informal exposition of the hierarchical attention. The formal mathematical derivation is deferred to the Appendix.

5.1 Hierarchical Structure As Inductive Bias

The error analysis in (Greengard and Rokhlin, 1987; Hackbusch, 1999) offers little direct insight since the attention matrix A in Eq. (3) is data dependent by definition and hence its analytical form like Eq. (11) and (12) is generally unknown. So gathering empirical evidences seems the only viable path to answer the first question listed above.

The ablation studies by (Khandelwal et al., 2018) examine the effect of context words on a language model. Within the context range of about 200 tokens, word order is only relevant within the 20 most recent tokens or about a sentence. In the long-range context, order has almost no effect on performance, suggesting that the model maintains a high-level, rough semantic representation of far-away words. The observation is succinctly summarized by the title of the paper "sharp nearby, fuzzy far away". Remarkably, this is in spirit very close to the key insight into the Multilevel Methods.

A few recent attention-related studies have explored this direction with some success, such as word-level and sentence-level attentions in (Miculicich et al., 2018; Abreu et al., 2019), and sentence-level and paragraph-level attentions in (Liu and Lapata, 2019). Even though the proposed hierarchical attention in these studies only has two levels, as opposed to ten or more levels typically used by the Multilevel Methods, the reported positive results are quite suggestive.

We therefore hypothesize that the same hierarchical low-rank structure as shown in Eq (13) might also hold for the attention matrix in many NLP tasks. And we treat it as the inductive bias in the hierarchical attention mechanism proposed in this paper. As pointed out in (Goyal and Bengio, 2020), inductive biases encourage the learning algorithm to prioritise solutions with certain properties. Hence good benchmark performance delivered by a Transformer-based model with proposed hierarchical attention can be regarded as a positive evidence to support the hierarchical low-rank structure hypothesis.

5.2 Informal Exposition of Hierarchical Attention

In the standard definition of attention in Eq. (3) and (4), there is no preference given to any keys based on the sequence distance between a query and keys. The observation in (Khandelwal et al.,

2018) clearly suggests that a distance-dependent attention mechanism should be a better alternative.

We will take three steps to informally explain the hierarchical attention mechanism. First, the attention matrix blocks for nearby, mid-distance and long-distance attention are separated in section 5.2.1. This is the first step toward the distance-dependent attention mentioned above. Second, a token hierarchy is established in section 5.2.2. Third, the hierarchical attention is constructed in section 5.2.3

5.2.1 Attention Partition

Consider a 16-word sentence in Fig. 1. The sentence is partitioned at three segment granularity. This induces a three-level partition of the attention matrix A for the original sequence:

$$A = A^{(2)} + A^{(1)} + A^{(0)} \tag{16}$$

where

$$A^{(2)} = \begin{bmatrix} 0 & A^{(2)}_{12} \\ A^{(2)}_{21} & 0 \end{bmatrix} \tag{17}$$

$$A^{(1)} = \begin{bmatrix} & A^{(1)}_{12} & & \\ A^{(1)}_{21} & & A^{(1)}_{23} & \\ & A^{(1)}_{32} & & A^{(1)}_{34} \\ & & A^{(1)}_{43} & \end{bmatrix} \tag{18}$$

$$A^{(0)} = \begin{bmatrix} A^{(0)}_{11} & A^{(0)}_{12} & & & \\ A^{(0)}_{21} & A^{(0)}_{22} & A^{(0)}_{23} & & \\ & \ddots & \ddots & \ddots & \\ & & & A^{(0)}_{87} & A^{(0)}_{88} \end{bmatrix}. \tag{19}$$

Note that the nonzero entries in $A^{(0)}$, $A^{(1)}$ and $A^{(2)}$ are the same as the corresponding entries of matrix A in Eq. (3). Matrix block size of $A^{(0)}_{ij}$, $A^{(1)}_{ij}$ and $A^{(2)}_{ij}$ is 2×2, 4×4 and 8×8, respectively. Following the key insight into Multilevel Methods, we perform no approximation to any level-0 matrix block $A^{(0)}_{ij}$ and apply a low-rank approximation to off-diagonal matrix blocks in $A^{(1)}$ and $A^{(2)}$. If we set the numerical rank of all these blocks to 2, then we can assemble the three rank

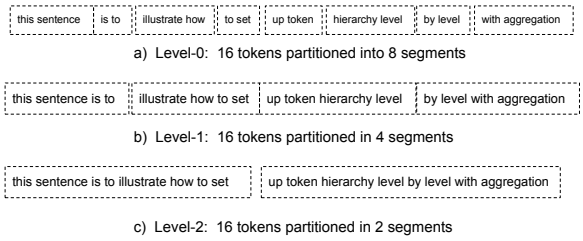

Figure 1: Token sequence partitions in three segment granularity.

maps into a single rank map as [4]

$$\begin{bmatrix} \begin{matrix} 2 & 2 \\ 2 & 2 \end{matrix} & 2 & 2 & \\ 2 & \begin{matrix} 2 & 2 \\ 2 & 2 \end{matrix} & & \\ & & \begin{matrix} 2 & 2 \\ 2 & 2 \end{matrix} & 2 \\ 2 & & 2 & \begin{matrix} 2 & 2 \\ 2 & 2 \end{matrix} \end{bmatrix}. \tag{20}$$

The hierarchical structure embodied by the predetermined rank map in Eq. (20) represents the inductive bias for the attention matrix A in Eq. (16). But this construction step is inefficient because we need to form the original attention matrix and then perform SVD to discover the low-rank approximation.

5.2.2 Token Hierarchy

To illustrate the notion of token hierarchy, consider the same 16-word sentence in Fig. 2. A simple 3-level binary-tree hierarchy can be set up by following the simple coarsening defined in Eq. (14): 1) At level-0, each one of the 16 words is mapped to its word embedding; 2) At level-1, each token (parent node) corresponds to a pair of adjacent words at level-0 (child nodes), which are shown inside each box. The embedding of each parent token is simply the average of its child token embeddings; 3) At level-2, each token (parent node) corresponds to one pair of adjacent tokens at level-1 (child nodes) or 4 adjacent words at level-0 (grand child nodes), which are shown inside each box. The embedding of each parent token is simply the average of its child token embeddings.

In general, the height of the binary tree is $O(log_2(L)$ and the total number of tree nodes is $O(2L)$, where L is the sequence length. We only need word embeddings for the leaf nodes since the

[4] We omit some of implementation details to handle the overlapping entries between adjacent levels.

embeddings of all other tree nodes can be recursively computed. The formal definition and notations of the recursion for query and key are detailed in section 6.1.

5.2.3 Informal Construction of Hierarchical Attention

It is clear from Fig. 2 that the embeddings of higher level tokens represent a coarser level representation of a larger chunk of the text. The tokens at different levels can be understood as multi-scale snapshots of the original token sequence at level-0. Hence this token hierarchy naturally induces a set of multi-scale attention matrices. Let $\tilde{A}^{(i)}$ be the attention matrix induced by the tokens at level-i. It is clear from Fig. 2 that the size of $\tilde{A}^{(0)}$, $\tilde{A}^{(1)}$ and $\tilde{A}^{(2)}$ is 16×16, 8×8 and 4×4, respectively. This multi-scale viewpoint does not directly lead to a useful algorithm since matrix $\tilde{A}^{(0)}$ contains all the information and there is little additional information from $\tilde{A}^{(1)}$ and $\tilde{A}^{(2)}$.

A key step to arrive at the hierarchical attention is to apply the contextual sliding window at each hierarchy level. The tokens at each level are partitioned into segments of size 2 in Fig. 2. One way to implement the local attention is to allow each query token segment to attend only two adjacent key token segments, one to its left and another to its right. At level-0, each query token segment also attends to the collocated key token segment. The token segment partition and local attention lead to a tri-diagonal block sparse matrix structure for $\tilde{A}^{(0)}$ and bi-diagonal block sparse matrix structure for $\tilde{A}^{(1)}$ and $\tilde{A}^{(2)}$. Their sparsity patterns are

$$\tilde{A}^{(0)} \propto \begin{bmatrix} 2 & 2 & & & & & & \\ 2 & 2 & 2 & & & & & \\ & 2 & 2 & 2 & & & & \\ & & 2 & 2 & 2 & & & \\ & & & 2 & 2 & 2 & & \\ & & & & 2 & 2 & 2 & \\ & & & & & 2 & 2 & 2 \\ & & & & & & 2 & 2 \end{bmatrix} \tag{21}$$

$$\tilde{A}^{(1)} \propto \begin{bmatrix} & 2 & & \\ 2 & & 2 & \\ & 2 & & 2 \\ & & 2 & \end{bmatrix} \tag{22}$$

$$\tilde{A}^{(2)} \propto \begin{bmatrix} & 2 \\ 2 & \end{bmatrix} \tag{23}$$

where the 2 in the nonzero blocks indicates that these are dense blocks of size 2×2.

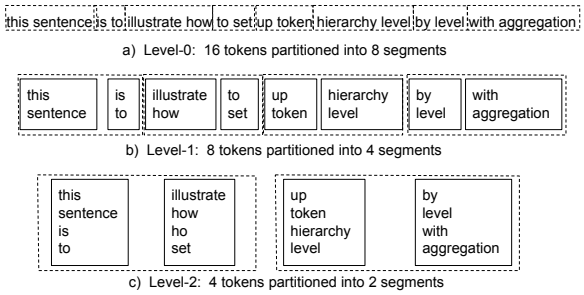

Figure 2: A three-level token hierarchy. Dashed boxes represent segmentation and solid boxes represents tokens.

It is clear that $\tilde{A}^{(0)}$ is identical to $A^{(0)}$ in Eq. (19). The efficiency gain comes from $\tilde{A}^{(2)}$ and $\tilde{A}^{(1)}$. Each nonzero entry in $\tilde{A}^{(2)}$ and $\tilde{A}^{(1)}$ captures the aggregated or coarse attention between two disjoint chunk of four and two tokens, respectively. Progressively larger token chunks lead to progressively lower-precision approximation to the original attention blocks. This is precisely the intention of the rank map in Eq. (20). We can now see that $\tilde{A}^{(2)}$ and $\tilde{A}^{(1)}$ provide an efficient way to approximate $A^{(2)}$ in Eq. (17) and $A^{(1)}$ in Eq. (18), respectively.

6 Key Components in Hierarchical Attention

6.1 Constructing Hierarchical Attention

The simple example in Fig. 2 can be easily generalized. Eq. (14) is used to coarsen or merge rows in matrices Q, K and V in Eq. (1). For sequence length $L = 2^{M+1}$, the coarsening establishes a binary tree of depth M for Q, K and V, respectively. Each tree node represents a matrix row and there are 2^{M+1-l} nodes or rows at level-l. To facilitate the discussion, we define a few hierarchy related notations here. Let $\tilde{Q}^{(l)}$, $\tilde{K}^{(l)}$ and $\tilde{V}^{(l)}$ be coarsened versions of Q, K and V at level-l in the binary tree. We note that $l = 0$ is a special case, which is defined as

$$\tilde{Q}^{(0)} = Q, \quad \tilde{K}^{(0)} = K, \quad \tilde{V}^{(0)} = V. \tag{24}$$

Following Eq. (14), the recursion to coarsen Q, K and V is:

$$\tilde{Q}_j^{(l+1)} = \frac{1}{2}(\tilde{Q}_{2j}^{(l)} + \tilde{Q}_{2j+1}^{(l)}) \tag{25}$$

$$\tilde{K}_j^{(l+1)} = \frac{1}{2}(\tilde{K}_{2j}^{(l)} + \tilde{K}_{2j+1}^{(l)}) \tag{26}$$

$$\tilde{V}_j^{(l+1)} = (\tilde{V}_{2j}^{(l)} + \tilde{V}_{2j+1}^{(l)}) \tag{27}$$

where $l = 0, 1, ..., M-2$ and $j = 0, 1, 2, ..., 2^{M-l}$. It should be noted that the coarsening of V in Eq. (27) does not have the averaging factor $\frac{1}{2}$. We defer more details on coarsening to Appendix Section A.1.

Now we are ready to compute the nonzero entries in Eq. (21), (22) and (23) and construct hierarchical attention matrix $\tilde{A}^{(l)}$. Substituting Eq. (25) and (26) into (4) and then into (3), we obtain

$$\tilde{A}^{(l)}_{ij} = e^{\tilde{S}^{(l)}_{ij}} = e^{\frac{\tilde{Q}^{(l)}_i (\tilde{K}^{(l)}_j)^T}{\sqrt{d}}} \tag{28}$$

Again, we note that $l = 0$ is a special case because $\tilde{A}^{(0)}_{ij} = A_{ij}$.

6.2 Applying Hierarchical Attention

The hierarchical matrix structure in Eq. (17), (18) and (19) naturally leads to a hierarchical approach to the matrix-matrix multiplication in Eq. (2) and the matrix-vector multiplication in Eq. (5). We use the matrix-matrix multiplication as an example since matrix-vector multiplication is just a special case of the matrix-matrix multiplication.

In view of Eq. (17), (18) and (19), we write the matrix-matrix multiplication in Eq. (2) as

$$Y = AV = Y^{(0)} + P^{(0)} \left(\tilde{Y}^{(1)} + P^{(1)} \tilde{Y}^{(2)} \right) \tag{29}$$

where

$$Y^{(0)} = A^{(0)} V^{(0)}, \ \tilde{Y}^{(l)} = \tilde{A}^{(l)} \tilde{V}^{(l)}, \ l = 1, 2 \tag{30}$$

We defer the detailed derivation of Eq. (29) to Appendix Section A.5 and A.6.

7 Algorithm And Computational Complexity

To facilitate the description and the complexity analysis of the algorithm, we define a few more hierarchy-related notations. In addition to sequence length L, number of hierarchy levels M and embedding or feature size d in Eq. (1), the new notations include: 1) N_r : numerical rank of the off-diagonal blocks (for instance, 2 in Eq. (20)). This is also the diagonal block size at level-0; 2) $N_b^{(l)}$: number of blocks at level-l. Note that L and d are usually data-dependent hyper-parameters, while N_r is the only model hyper-parameter responsible for our method's inductive bias. In turn, $N_b^{(l)}$ and M are derived parameters, computed as:

$$N_b^{(0)} = \frac{L}{N_r}, \ N_b^{(l+1)} = \frac{N_b^{(l)}}{2} \tag{31}$$

$$M = \log_2(N_b^{(0)}). \tag{32}$$

It is easy to verify that

$$\sum_{l=0}^{M-1} N_b^{(l)} = \sum_{l=0}^{M-1} \frac{N_b^{(0)}}{2^l} \approx 2N_b^{(0)}. \tag{33}$$

It is important to note that only the diagonal blocks at level-0 and the super-diagonal and sub-diagonal blocks at level-l are needed in applying the hierarchical attention matrix. This is clearly shown in Eq. (21)- (23). This means that only $N_b^{(l)} - 1$ super-diagonal and sub-diagonal blocks are computed at level-l. This is crucial to the overall linear complexity in run time and memory.

We should also note that all matrix blocks in coarse attention matrix $\tilde{A}^{(l)}$ have the same size $N_r \times N_r$. This is due to the rank map in Eq. (20). This is crucial for efficiency reason since the single-instruction-multiple-data (SIMD) programming style supported by the dense linear algebra libraries for GPU and TPU encourages uniform tensor shapes.

We summarize the main steps to construct and apply the hierarchical attention in Algorithm 1.

Algorithm 1 H-Transformer-1D

Input: Q**(query),** K**(key),** V**(value)**
Output: Z

Coarsen Q using Eq. (25) and coarsen K using Eq. (26)
Compute diagonal blocks in $\tilde{A}^{(0)}$ and super-diagonal and sub-diagonal blocks in $\tilde{A}^{(l)}$ using Eq. (28)
Coarsen V using Eq. (27)
Compute $Y = AV$ in Eq. (2) using Eq. (29)
Compute D in Eq. (5) using Eq. (29)
Compute $Z = D^{-1}Y$

The computational cost for Algorithm 1 has two parts:

1. Computing the hierarchical attention matrix:
   (a) diagonal blocks at level-0: $dN_r^2 N_b^{(0)}$
   (b) Super- and sub-diagonal blocks at level-l: $4dN_r^2(N_b^{(l)} - 1)$

(c) total: $5dLN_r = O(dL)$

2. Computing matrix-matrix (MM) multiplication in Eq. (2) and matrix-vector (MV) multiplication in Eq. (5):

   (a) MM: $5dLN_r$

   (b) MV: $5LN_r$

   (c) total: $5(d+1)LN_r = O(dL)$

So the overall run time complexity of the hierarchical attention algorithm is $O(dL)$. Likewise, the memory complexity can be shown to be $O(dL)$ as well. We defer the detailed analysis to appendix Section A.5 and A.6.

8 Experiments And Results

We have implemented the proposed hierarchical attention using Jax, an open source library [5] for automatic gradient computation and linear algebra operations on GPUs and TPUs. All numerical operations in our algorithm use the Numpy native linear algebra functions supported by Jax. In all our experiments in this section, we use the standard Transformer architecture described in (Vaswani et al., 2017) as the backbone for our H-Transformer-1D model. Unless specified otherwise, the model parameters are: number of layers is 6, number of heads is 8, word embedding size is 512 and the feed-forward module (FFN) size is 2048. We follow the API for the standard multihead scaled dot-product attention implementation [6] so that we can perform a simple drop-in replacement of the standard multihead attention with our hierarchical attention implementation. This allows for an easy and fair comparison.

8.1 Long-Range Arena

The open-source Long-Range Arena (LRA) benchmark [7] has been proposed as a standard way to probe and quantify the capabilities of various xformer (long-range Transformer) architectures (Tay et al., 2020c). In our case, it also serves to highlight the effectiveness of the inductive bias inspired by the H-Matrix method, as well as the capability of our hierarchical attention to handle long sequences.

The LRA has several desirable qualities that made us focus on it as a primary evaluation benchmark: **generality** (restricted to encoder-only tasks to accommodate most proposals); **simplicity** (no pretraining, no data augmentation allowed); **difficulty** (large headroom with existing approaches); **long-input focus** (so that modeling improvements in this area are visible); **diverse** (6 tasks, covering math, language, image, and spatial modeling); and **lightweight** (so that modeling improvements are measurable independently of the ability to train and run high-capacity models).

The tasks that comprise LRA are: **ListOps** (sequences of arithmetical expressions of lengths of up to 2K that tests the ability to reason hierarchically while handling long context); **Text** (byte/character-level text classification at document level, which both simulates longer input sequences – max length 4K – and increases the difficulty level); **Retrieval** (byte/character-level document retrieval, which simulates the ability to model document similarity as a score between two independently-encoded long input sequences – max length 4K + 4K = 8K); **Image** (image classification based on the CIFAR-10 dataset, where an NxN image is flattened to a sequence of length N^2 pixels); **Pathfinder** (long-range spatial dependency task, with images consisting of two small circles and dash-line paths that either connect the two circles or not – image dimensions of 32x32 for a pixel sequence of length 1,024); **Path-X** (same as Pathfinder, but for image dimensions of 128x128 for a total pixel sequence of length 16,384). The default Transformer model parameters such as number of layers and number of heads etc are pre-determined by the benchmark configuration for each task.

The results obtained by our H-Transformer-1D model on the LRA benchmark are given in Table 1. Overall, the H-Transformer-1D model achieves 61.41 average accuracy, a +6.4 points improvement over the previous-best average performance from BigBird (Zaheer et al., 2020). We want to highlight ListOps, Text and Retrieval because they all involve long sequences and H-Transformer-1D model improves SOTA performance by relatively large margins. These should be strong evidences to support our hypothesis in section 5.1 and validate the inductive bias due to the hierarchical attention.

[5] https://github.com/google/jax

[6] https://github.com/google/flax/blob/master/flax/nn

[7] https://github.com/google-research/long-range-arena

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Chance	10.00	50.00	50.00	10.00	50.00	50.00	44.00
Transformer	36.37	64.27	57.46	42.44	71.40	FAIL	54.39
Local Attention	15.82	52.98	53.39	41.46	66.63	FAIL	46.06
Sparse Trans.	17.07	63.58	<u>59.59</u>	<u>44.24</u>	71.71	FAIL	51.24
Longformer	35.63	62.85	56.89	42.22	69.71	FAIL	53.46
Linformer	35.70	53.94	52.27	38.56	<u>76.34</u>	FAIL	51.36
Reformer	<u>37.27</u>	56.10	53.40	38.07	68.50	FAIL	50.67
Sinkhorn Trans.	33.67	61.20	53.83	41.23	67.45	FAIL	51.39
Synthesizer	36.99	61.68	54.67	41.61	69.45	FAIL	52.88
BigBird	36.05	64.02	59.29	40.83	74.87	FAIL	<u>55.01</u>
Linear Trans.	16.13	<u>65.90</u>	53.09	42.34	75.30	FAIL	50.55
Performer	18.01	65.40	53.82	42.77	**77.05**	FAIL	51.41
H-Transformer-1D	**49.53**	**78.69**	**63.99**	**46.05**	68.78	FAIL	**61.41**

Table 1: Experimental results on long-range arena benchmark. Best model is in boldface and second best is underlined. All models do not learn anything on Path-X task, contrary to the Pathfinder task and this is denoted by FAIL. Path-X is not counted toward the Average score as it has no impact on relative performance.

Model	perplexity	parameters
(Dai et al., 2019)	21.8	800M
(Baevski and Auli, 2019)	23.02	1000M
(Dai et al., 2019)	23.5	465M
(Baevski and Auli, 2019)	23.91	465M
(Shazeer et al., 2018)	24.0	4900M
Transformer baseline	30.04	53M
Transformer baseline	24.8	144M
H-Transformer-1D $N_r = 16$	23.95	53M
H-Transformer-1D $N_r = 16$	**20.25**	144M

Table 2: Experimental results on one-billion word benchmark. We compare previous SOTA results obtained with models of size 465M-4900M parameters against the performance of the quadratic attention baseline and the H-Transformer-1D models.

8.2 Language Models Trained on One-Billion Words

We have used Flax, an open-source library [8] to train neural networks, as the code base for the model training. Our H-Transformer-1D model uses the standard Transformer decoder implementation in Flax as the backbone. Only the attention is replaced with our hierarchical attention. We trained both the Transformer baseline and H-Transformer-1D on the One-Billion Word benchmark (Chelba et al., 2014). We tried different N_r (numerical rank) in our H-Transformer-1D model. These represent different inductive bias. We found that H-Transformer-1D with $N_r = 16$ generated text with quality comparable to that of the baseline Transformer. For both Transformer baseline and H-Transformer-1D, we also tried two sets of model parameters: 1) embedding size is 512 and feed-forward module size is 2048 and hence the parameter count is 53M; 2) embedding size is 1024 and feed-forward module size is 4096 and hence the parameter count is 144M. The test perplexity results of these four models and various SOTA models are shown in table 2.

H-Transformer-1D delivers the lowest perplexity to-date while using $5\times$ smaller model capacity than that of the previous SOTA model Transformer-XL (Dai et al., 2019). This is another strong evidence to support our hypothesis in section 5.1 and validate the inductive bias due to the

[8] https://github.com/google/flax

hierarchical attention.

9 Conclusions and Future Work

We have proposed a new Transformer attention using the inductive bias inspired by the H-Matrix. The new algorithm has linear complexity in run time and memory usage and is fully compatible with dense linear algebra libraries on GPU and TPU. The effectiveness of this new attention is demonstrated by the empirical evidences from long-range arena benchmark and One-Billion word language modeling. Future work include applying the new attention to music and genomics, developing proper inductive bias for cross-attention and extending to 2D images.

References

Jader Abreu, Luis Fred, David Macêdo, and C. Zanchettin. 2019. Hierarchical attentional hybrid neural networks for document classification. *ArXiv*, abs/1901.06610.

Joshua Ainslie, S. Ontañón, C. Alberti, V. Cvicek, Zachary Kenneth Fisher, Philip Pham, Anirudh Ravula, S. Sanghai, Qifan Wang, and L. Yang. 2020. Etc: Encoding long and structured inputs in transformers. In *EMNLP*.

Alexei Baevski and M. Auli. 2019. Adaptive input representations for neural language modeling. *ArXiv*, abs/1809.10853.

I. Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V. Le. 2019. Attention augmented convolutional networks. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3285–3294.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *ArXiv*, abs/2004.05150.

A. Brandt and A. A. Lubrecht. 1990. Multilevel matrix multiplication and fast solution of integral equations. 90:348–370.

W.L. Briggs, V.E. Henson, and S.F. McCormick. 2000. *A Multigrid Tutorial*. SIAM.

Tom B. Brown, Benjamin Pickman Mann, Nick Ryder, Melanie Subbiah, Jean Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, G. Krüger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric J Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *ArXiv*, abs/2005.14165.

Ciprian Chelba, Tomas Mikolov, M. Schuster, Qi Ge, T. Brants, Phillipp Koehn, and T. Robinson. 2014. One billion word benchmark for measuring progress in statistical language modeling. *ArXiv*, abs/1312.3005.

Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. 2020. Generative pretraining from pixels. *Proceedings of the 37th International Conference on Machine Learning*, PMLR 119.

R. Child, Scott Gray, A. Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *ArXiv*, abs/1904.10509.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Jared Davis, Tamás Sarlós, David Belanger, Lucy J. Colwell, and Adrian Weller. 2020. Masked language modeling for proteins via linearly scalable long-context transformers. *ArXiv*, abs/2006.03555.

Zihang Dai, Z. Yang, Yiming Yang, J. Carbonell, Quoc V. Le, and R. Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. In *ACL*.

J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

G.H. Golub and C.F. Van Loan. 1996. *Matrix Computation*. The John Hopkins University Press, Baltimore.

Anirudh Goyal and Yoshua Bengio. 2020. Inductive biases for deep learning of higher-level cognition. *ArXiv*, abs/2011.15091.

L Greengard. 1994. Fast algorithms for classical physics. *Science*, 265:909–914.

L Greengard and V Rokhlin. 1987. A fast algorithm for particle simulations. 73:325–348.

W. Hackbusch. 1999. A sparse matrix arithmetic based on h-matrices. part I: Introduction to H-matrices. *Computing*, 62:89–108.

W. Hackbusch. 2000. A sparse matrix arithmetic based on H-matrices. part II: Application to multi-dimensional problems. *Computing*, 64:21–47.

Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. 2019. Axial attention in multidimensional transformers. *ArXiv*, abs/1912.12180.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. 2018. Music transformer. *arXiv: Learning*.

S. Kapur and D.E. Long. 1997. IES3: A fast integral equation solver for efficient 3-dimensional extraction. *International Conference on Computer Aided-Design*, pages 448–455.

Urvashi Khandelwal, He He, Peng Qi, and Dan Jurafsky. 2018. Sharp nearby, fuzzy far away: How neural language models use context. *ArXiv*, abs/1805.04623.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *ArXiv*, abs/2001.04451.

Yang Liu and Mirella Lapata. 2019. Hierarchical transformers for multi-document summarization. In *ACL*.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. *ArXiv*, abs/1508.04025.

Chris Manning and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA.

Lesly Miculicich, Dhananjay Ram, Nikolaos Pappas, and James Henderson. 2018. Document-level neural machine translation with hierarchical attention networks. In *EMNLP*.

K. Nabors, T. Korsmeyer, and J. White. 1994. Multipole accelerated preconditioned iterative methods for three-dimensional potential integral equations of the first kind. *SIAM J. Sci. and Stat. Comp.*

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer. *ArXiv*, abs/1802.05751.

Joel R. Phillips and J. K. White. 1997. A precorrected-FFT method for electrostatic analysis of complicated 3D structures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1059–1072.

Jiezhong Qiu, Hao Ma, Omer Levy, Scott Yih, Sinong Wang, and Jie Tang. 2019. Blockwise self-attention for long document understanding. *ArXiv*, abs/1911.02972.

Prajit Ramachandran, Niki Parmar, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jonathon Shlens. 2019. Stand-alone self-attention in vision models. *ArXiv*, abs/1906.05909.

Aurko Roy, M. Saffar, Ashish Vaswani, and David Grangier. 2020. Efficient content-based sparse attention with routing transformers. *ArXiv*, abs/2003.05997.

Noam Shazeer, Youlong Cheng, Niki Parmar, Dustin Tran, Ashish Vaswani, Penporn Koanantakool, P. Hawkins, H. Lee, Mingsheng Hong, C. Young, Ryan Sepassi, and Blake A. Hechtman. 2018. Mesh-tensorflow: Deep learning for supercomputers. In *NeurIPS*.

W. Shi, J. Liu, N. Kakani, and T. Yu. 1998. A fast hierarchical algorithm for 3-d capacitance extraction. *ACM/IEEE Design Automation Conference*.

Yi Tay, Dara Bahri, Donald Metzler, D. Juan, Zhe Zhao, and Che Zheng. 2020a. Synthesizer: Rethinking self-attention in transformer models. *ArXiv*, abs/2005.00743.

Yi Tay, Dara Bahri, L. Yang, Donald Metzler, and D. Juan. 2020b. Sparse sinkhorn attention. In *ICML*.

Yi Tay, M. Dehghani, Samira Abnar, Y. Shen, Dara Bahri, Philip Pham, J. Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. 2020c. Long range arena: A benchmark for efficient transformers. *ArXiv*, abs/2011.04006.

Yi Tay, M. Dehghani, Dara Bahri, and Donald Metzler. 2020d. Efficient transformers: A survey. *ArXiv*, abs/2009.06732.

L.N. Trefethen and D. Bau. 1997. *Numerical linear algebra*. SIAM, Philadelphia.

Ulrich Trottenberg, Cornelius W. Oosterlee, and Anton Schuller. 2000. *Multigrid*. Academic Press.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *ArXiv*, abs/1706.03762.

Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. *ArXiv*, abs/1710.10903.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity. *ArXiv*, abs/2006.04768.

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2020. Big bird: Transformers for longer sequences.

Hao-Yi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2020. Informer: Beyond efficient transformer for long sequence time-series forecasting. *ArXiv*, abs/2012.07436.

Zhenhai Zhu, Ben Song, and J. K. White. 2005. Algorithms in FastImp: A fast and wideband impedance extraction program for complicated 3D geometries. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Zhenhai Zhu and J. K. White. 2005. Fastsies: a fast stochastic integral equation solver for modeling the rough surface effect. *International Conference on Computer Aided-Design*, pages 675–682.

A Appendix

A.1 Restriction or Coarsening Matrices

For sequence length $L = 2^{M+1}$, the coarsening establishes a binary tree of depth M for Q, K and V, respectively. The root of the binary tree at level-$(M-1)$ has two nodes which correspond to the two matrix rows coarsened from four matrix rows at level-$(M-2)$. The piecewise constant restriction matrix at level-$(M-2)$ is

$$R^{(M-2)} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}_{2\times 4}. \tag{34}$$

Likewise, the piecewise constant restriction matrix at level-$(M-3)$ is

$$\begin{aligned} R^{(M-3)} &= \left[\begin{array}{cccc|cccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}\right]_{4\times 8} \\ &= \left[\begin{array}{c|c} R^{(M-2)} & 0 \\ \hline 0 & R^{(M-2)} \end{array}\right]. \end{aligned} \tag{35}$$

In general, the restriction matrices follow the recursion

$$R^{(l-1)} = \left[\begin{array}{c|c} R^{(l)} & 0 \\ \hline 0 & R^{(l)} \end{array}\right] \tag{36}$$

which starts from $R^{(M-2)}$ of size 2×4 and goes backward to $R^{(0)}$ of size $\frac{L}{2} \times L$.

A.2 Interpolation Matrices

Given $Y^{(l)}$ at level-l, the interpolated $Y^{(l-1)}$ at level-$(l-1)$ can be written as

$$Y^{(l-1)} = P^{(l)} Y^{(l)} \tag{37}$$

where $l = 1, 2, ..., M-1$, sparse matrix $P^{(l)}$ has size $L^{(l-1)} \times L^{(l)}$, and $L^{(l)} = 2^{M-l}$ is the node count at level-l of the binary tree.

This recursion also follows the binary tree hierarchy. The four matrix rows at level-$(M-2)$ are interpolated from the two matrix rows at level-$(M-1)$. Specifically, the piecewise constant interpolation matrix at level-$(M-1)$ is

$$P^{(M-1)} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}_{4\times 2}. \tag{38}$$

Likewise, the piecewise constant interpolation matrix at level-$(M-2)$ is

$$\begin{aligned} P^{(M-2)} &= \left[\begin{array}{cc|cc} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{array}\right]_{8\times 4} \\ &= \left[\begin{array}{c|c} P^{(M-1)} & 0 \\ \hline 0 & P^{(M-1)} \end{array}\right]. \end{aligned} \tag{39}$$

In general, the interpolation matrices follow the recursion

$$P^{(l-1)} = \left[\begin{array}{c|c} P^{(l)} & 0 \\ \hline 0 & P^{(l)} \end{array}\right] \tag{40}$$

which starts from $P^{(M-1)}$ of size 4×2 and goes backward to $P^{(0)}$ of size $L \times \frac{L}{2}$. In view of Eq. (34) and (38), it is obvious that

$$P^{(M-1)} = (R^{(M-2)})^T. \tag{41}$$

In view of the recursions in Eq. (36) and (40), it is easy to prove by induction that

$$P^{(l)} = (R^{(l-1)})^T. \tag{42}$$

A.3 Expansion Matrices

For the purpose of factored low-rank approximation for the off-diagonal attention matrix blocks, we design a series of so-called expansion matrices. The first two expansion matrices in this series are

$$\begin{aligned} T^{(M-1)} &= P^{(M-1)} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}_{4\times 2} \\ &= \begin{bmatrix} \mathbf{1}_2 & 0 \\ 0 & \mathbf{1}_2 \end{bmatrix} \end{aligned} \tag{43}$$

and

$$\begin{aligned} T^{(M-2)} &= P^{(M-2)} P^{(M-1)} = \left[\begin{array}{c|c} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ \hline 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{array}\right]_{8\times 2} \\ &= \begin{bmatrix} \mathbf{1}_4 & 0 \\ 0 & \mathbf{1}_4 \end{bmatrix} \end{aligned} \tag{44}$$

where $\mathbf{1}_N$ is a length-N vector of ones. The general form of matrix $T^{(l)}$ is defined as

$$T^{(l)} = \Pi_{i=l}^{M-1} P^{(i)} \tag{45}$$

where $l = 1, 2, ..., M-1$. In view of Eq. (43), (45) and (40), it is easy to prove by induction that

$$T^{(l)} = \begin{bmatrix} \mathbf{1}_{2^{M-l}} & 0 \\ 0 & \mathbf{1}_{2^{M-l}} \end{bmatrix} \tag{46}$$

and it has size $2^{M-l+1} \times 2$. Further more, in view of Eq. (45) and (42), we have

$$(T^{(l)})^T = \Pi_{i=M-1}^{l} R^{(i-1)}. \tag{47}$$

A.4 Low-Rank Factored Form

Matrix $T^{(l)}$ plays a pivotal role in constructing the low-rank approximation to the off-diagonal attention matrix blocks. Let the ij-th block in the coarsened attention matrix at level-1 be

$$\tilde{A}_{ij}^{(1)} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \tag{48}$$

where a_{ij} is the entry resulted from the inner product between a row in $\tilde{Q}^{(1)}$ and $\tilde{K}^{(1)}$. The rank-2 approximation to the corresponding ij-th block in the original attention matrix A at level-1 can be written as

$$\begin{aligned} A_{ij}^{(1)} &\approx T^{(M-1)} \tilde{A}_{ij}^{(1)} (T^{(M-1)})^T \qquad (49) \\ &= \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \\ &= \left[\begin{array}{cc|cc} a_{11} & a_{11} & a_{12} & a_{12} \\ a_{11} & a_{11} & a_{12} & a_{12} \\ \hline a_{21} & a_{21} & a_{22} & a_{22} \\ a_{21} & a_{21} & a_{22} & a_{22} \end{array}\right]. \qquad (50) \end{aligned}$$

It is clear that the resulting 4×4 matrix $A_{ij}^{(1)}$ is essentially the piecewise constant interpolation of the 2×2 matrix $\tilde{A}_{ij}^{(1)}$ along row and column direction. And since both $T^{(M-1)}$ and $\tilde{A}_{ij}^{(1)}$ have full rank 2, $A_{ij}^{(1)}$ necessarily has rank 2. One can also view a_{ij} as being similar to the average value at the ij-th cluster center in the K-mean method. The role of matrix $T^{(M-1)}$ is to expand from these 2×2 clusters to the 4×4 grid and hence the name expansion matrix.

Since we maintain the same numerical rank 2 for all super- and sub-diagonal attention matrix blocks, the rank-2 approximation to the ij-th block in the original attention matrix A at level-l is

$$\begin{aligned} A_{ij}^{(l)} &\approx T^{(M-l)} \tilde{A}_{ij}^{(l)} (T^{(M-l)})^T \\ &= \Pi_{i=M-l}^{M-1} P^{(i)} \tilde{A}_{ij}^{(l)} \Pi_{i=M-1}^{M-l} R^{(i-1)} \end{aligned} \tag{51}$$

where the last equality is due to Eq. (45) and (47).

We note that matrix $T^{(l)}$ has full column rank 2 by design and this can be easily shown from Eq. (46). We have used this fact to construct the rank-2 approximation in Eq. (51).

A.5 Construct Hierarchical Attention Matrix

To see how Eq. (51) can be used, consider a simple three-level partition of the attention matrix A for sequence length $L = 16$

$$A = \left[\begin{array}{c|c} A_{11}^{(2)} & A_{12}^{(2)} \\ \hline A_{21}^{(2)} & A_{22}^{(2)} \end{array}\right] \tag{52}$$

$$A_{11}^{(2)} = \left[\begin{array}{cc|cc} A_{11}^{(0)} & A_{12}^{(0)} & \multicolumn{2}{c}{\multirow{2}{*}{$A_{12}^{(1)}$}} \\ A_{21}^{(0)} & A_{22}^{(0)} & & \\ \hline \multicolumn{2}{c|}{\multirow{2}{*}{$A_{21}^{(1)}$}} & A_{33}^{(0)} & A_{34}^{(0)} \\ & & A_{43}^{(0)} & A_{44}^{(0)} \end{array}\right] \tag{53}$$

$$A_{22}^{(2)} = \left[\begin{array}{cc|cc} A_{55}^{(0)} & A_{56}^{(0)} & \multicolumn{2}{c}{\multirow{2}{*}{$A_{34}^{(1)}$}} \\ A_{65}^{(0)} & A_{66}^{(0)} & & \\ \hline \multicolumn{2}{c|}{\multirow{2}{*}{$A_{43}^{(1)}$}} & A_{77}^{(0)} & A_{78}^{(0)} \\ & & A_{87}^{(0)} & A_{88}^{(0)} \end{array}\right] \tag{54}$$

where the size of level-0, level-1 and level-2 matrix blocks is 2×2, 4×4 and 8×8, respectively. Note that the number of levels is $M = log_2(L/2) = 3$. We use this simple three-level example to illustrate the key steps in both constructing and applying the hierarchical attention matrix.

In view of Eq. (51), we have

$$A \approx \left[\begin{array}{c|c} \tilde{A}_{11}^{(2)} & T^{(1)} \tilde{A}_{12}^{(2)} (T^{(1)})^T \\ \hline T^{(1)} \tilde{A}_{21}^{(2)} (T^{(1)})^T & \tilde{A}_{22}^{(2)} \end{array}\right] \tag{55}$$

$$\tilde{A}_{11}^{(2)} = \left[\begin{array}{cc|c} A_{11}^{(0)} & A_{12}^{(0)} & \multirow{2}{*}{$T^{(2)} \tilde{A}_{12}^{(1)} (T^{(2)})^T$} \\ A_{21}^{(0)} & A_{22}^{(0)} & \\ \hline \multicolumn{2}{c|}{\multirow{2}{*}{$T^{(2)} \tilde{A}_{21}^{(1)} (T^{(2)})^T$}} & \begin{array}{cc} A_{33}^{(0)} & A_{34}^{(0)} \\ A_{43}^{(0)} & A_{44}^{(0)} \end{array} \\ & & \end{array}\right] \tag{56}$$

$$\tilde{A}_{22}^{(2)} = \left[\begin{array}{cc|cc} A_{55}^{(0)} & A_{56}^{(0)} & \multicolumn{2}{c}{\multirow{2}{*}{$T^{(2)}\tilde{A}_{34}^{(1)}(T^{(2)})^T$}} \\ A_{65}^{(0)} & A_{66}^{(0)} & & \\ \hline \multicolumn{2}{c|}{\multirow{2}{*}{$T^{(2)}\tilde{A}_{43}^{(1)}(T^{(2)})^T$}} & A_{77}^{(0)} & A_{78}^{(0)} \\ & & A_{87}^{(0)} & A_{88}^{(0)} \end{array}\right]. \tag{57}$$

We note that matrices $T^{(l)}, l = 1, 2$ are never explicitly formed and are only implicitly used, as shown in next section. So only the diagonal blocks at level-0 and super- and sub-diagonal blocks of the coarsened matrix $\tilde{A}$ at level-l need to be explicitly computed. By design, all these blocks have the same size 2×2 if we set the numerical rank to $N_r = 2$. The total number of super- and sub-diagonal blocks in the binary tree hierarchy is upper bounded by twice the number of super- and sub-diagonal blocks at level-0, which is $2N_b^{(0)}$. Hence the total number of entries is $5N_b^{(0)}N_r^2 = 5LN_r = O(LN_r)$. Each entry is equal to the inner product between $\tilde{Q}_i^{(l)}$ and $\tilde{K}_j^{(l)}$ and hence the run time cost per entry is $O(d)$, where d is the embedding size. So the final total run time cost is $O(Ld)$ and memory foot print is $O(L)$. Here we leave out N_r since it is a constant model hyper parameter.

A.6 Apply Hierarchical Attention Matrix

Computing matrix-matrix product AV follows the hierarchical structure of matrix A in Eq. (55), (56) and (57). We first partition matrix V according to the three-level binary tree established by the coarsening process, i.e.,

$$V = \left[\begin{array}{c} V_1^{(0)} \\ V_2^{(0)} \\ \hline \vdots \\ \hline V_7^{(0)} \\ V_8^{(0)} \end{array}\right] = \left[\begin{array}{c} V_1^{(1)} \\ V_2^{(1)} \\ \hline V_3^{(1)} \\ V_4^{(1)} \end{array}\right] = \left[\begin{array}{c} V_1^{(2)} \\ V_2^{(2)} \end{array}\right]. \tag{58}$$

Note that these are partitions of the same matrix V at 3 different levels. For sequence length $L = 16$, matrix V has size $16 \times d$, and the size of the partitioned blocks $V_i^{(0)}$, $V_j^{(1)}$ and $V_k^{(2)}$ are $2 \times d$, $4 \times d$ and $8 \times d$, respectively. In the derivation to come, we may exchange partitions at different levels. For instance, in view of Eq. (58), we have

$$V_1^{(2)} = \left[\begin{array}{c} V_1^{(1)} \\ V_2^{(1)} \end{array}\right]. \tag{59}$$

So we may replace $V_1^{(2)}$ with the right-hand side in Eq. (59).

In view of Eq. (52) and (58), matrix-matrix product AV can be written as

$$\begin{aligned} Y &= AV = \left[\begin{array}{c} A_{11}^{(2)}V_1^{(2)} \\ A_{22}^{(2)}V_2^{(2)} \end{array}\right] + \left[\begin{array}{c} A_{12}^{(2)}V_2^{(2)} \\ A_{21}^{(2)}V_1^{(2)} \end{array}\right] \\ &= \left[\begin{array}{c} A_{11}^{(2)}V_1^{(2)} \\ A_{22}^{(2)}V_2^{(2)} \end{array}\right] + Y^{(2)}. \end{aligned} \tag{60}$$

In view of Eq. (55), we have

$$\begin{aligned} Y^{(2)} &= \left[\begin{array}{c} A_{12}^{(2)}V_2^{(2)} \\ A_{21}^{(2)}V_1^{(2)} \end{array}\right] \\ &\approx \left[\begin{array}{c} T^{(1)}\tilde{A}_{12}^{(2)}(T^{(1)})^T V_2^{(2)} \\ T^{(1)}\tilde{A}_{21}^{(2)}(T^{(1)})^T V_1^{(2)} \end{array}\right] \\ &= \left[\begin{array}{c} P^{(1)}P^{(2)}\tilde{A}_{12}^{(2)}R^{(1)}R^{(0)}V_2^{(2)} \\ P^{(1)}P^{(2)}\tilde{A}_{21}^{(2)}R^{(1)}R^{(0)}V_1^{(2)} \end{array}\right] \\ &= P^{(0)}P^{(1)}\left[\begin{array}{c} \tilde{A}_{12}^{(2)}\tilde{V}_2^{(2)} \\ \tilde{A}_{21}^{(2)}\tilde{V}_1^{(2)} \end{array}\right] \\ &= P^{(0)}P^{(1)}\left[\begin{array}{c} \tilde{Y}_1^{(2)} \\ \tilde{Y}_2^{(2)} \end{array}\right] \end{aligned} \tag{61}$$

where

$$\left[\begin{array}{c} \tilde{V}_1^{(2)} \\ \tilde{V}_2^{(2)} \end{array}\right] = \left[\begin{array}{c} R^{(1)}R^{(0)}V_1^{(2)} \\ R^{(1)}R^{(0)}V_2^{(2)} \end{array}\right]. \tag{62}$$

The third equality in Eq. (61) is due to Eq. (45) and (47) where $l = 1$. The fourth equality in Eq. (61) is due to Eq. (40).

In view of Eq. (56), we have

$$\begin{aligned} & A_{11}^{(2)}V_1^{(2)} \approx \tilde{A}_{11}^{(2)}V_1^{(2)} \\ &= \left[\begin{array}{cc|cc} A_{11}^{(0)} & A_{12}^{(0)} & \multicolumn{2}{c}{\multirow{2}{*}{$T^{(2)}\tilde{A}_{12}^{(1)}(T^{(2)})^T$}} \\ A_{21}^{(0)} & A_{22}^{(0)} & & \\ \hline \multicolumn{2}{c|}{\multirow{2}{*}{$T^{(2)}\tilde{A}_{21}^{(1)}(T^{(2)})^T$}} & A_{33}^{(0)} & A_{34}^{(0)} \\ & & A_{43}^{(0)} & A_{44}^{(0)} \end{array}\right] V_1^{(2)} \\ &= \left[\begin{array}{c} Y_1^{(0)} \\ Y_2^{(0)} \\ Y_3^{(0)} \\ Y_4^{(0)} \end{array}\right] + Y_1^{(1)} \end{aligned} \tag{63}$$

where

$$
\begin{aligned}
Y_1^{(1)} &= \begin{bmatrix} T^{(2)}\tilde{A}_{12}^{(1)}(T^{(2)})^T V_2^{(1)} \\ T^{(2)}\tilde{A}_{21}^{(1)}(T^{(2)})^T V_1^{(1)} \end{bmatrix} \\
&= \begin{bmatrix} P^{(2)}\tilde{A}_{12}^{(1)} R^{(1)} V_2^{(1)} \\ P^{(2)}\tilde{A}_{21}^{(1)} R^{(1)} V_1^{(1)} \end{bmatrix} \\
&= P^{(1)} \begin{bmatrix} \tilde{A}_{12}^{(1)}\tilde{V}_2^{(1)} \\ \tilde{A}_{21}^{(1)}\tilde{V}_1^{(1)} \end{bmatrix} \\
&= P^{(1)} \begin{bmatrix} \tilde{Y}_1^{(1)} \\ \tilde{Y}_2^{(1)} \end{bmatrix} \qquad (64)
\end{aligned}
$$

and

$$
\begin{bmatrix} \tilde{V}_1^{(1)} \\ \tilde{V}_2^{(1)} \end{bmatrix} = \begin{bmatrix} R^{(1)} V_1^{(1)} \\ R^{(1)} V_2^{(1)} \end{bmatrix}. \qquad (65)
$$

The second equality in Eq. (64) is due to Eq. (45) and (47) where $l = 2$. The third equality in Eq. (64) is due to Eq. (40).

In view of Eq.(57), we have

$$
\begin{aligned}
& A_{22}^{(2)} V_2^{(2)} \approx \tilde{A}_{22}^{(2)} V_2^{(2)} \\
&= \left[\begin{array}{cc|c} A_{55}^{(0)} & A_{56}^{(0)} & T^{(1)}\tilde{A}_{34}^{(1)}(T^{(1)})^T \\ A_{65}^{(0)} & A_{66}^{(0)} & \\ \hline T^{(1)}\tilde{A}_{43}^{(1)}(T^{(1)})^T & & \begin{array}{cc} A_{77}^{(0)} & A_{78}^{(0)} \\ A_{87}^{(0)} & A_{88}^{(0)} \end{array} \end{array}\right] V_2^{(2)} \\
&= \begin{bmatrix} Y_5^{(0)} \\ Y_6^{(0)} \\ Y_7^{(0)} \\ Y_8^{(0)} \end{bmatrix} + Y_2^{(1)} \qquad (66)
\end{aligned}
$$

where

$$
\begin{aligned}
Y_2^{(1)} &= \begin{bmatrix} P^{(2)}\tilde{A}_{34}^{(1)} R^{(1)} V_4^{(1)} \\ P^{(2)}\tilde{A}_{43}^{(1)} R^{(1)} V_3^{(1)} \end{bmatrix} \\
&= P^{(1)} \begin{bmatrix} \tilde{A}_{34}^{(1)}\tilde{V}_4^{(1)} \\ \tilde{A}_{43}^{(1)}\tilde{V}_3^{(1)} \end{bmatrix} \\
&= P^{(1)} \begin{bmatrix} \tilde{Y}_3^{(1)} \\ \tilde{Y}_4^{(1)} \end{bmatrix} \qquad (67)
\end{aligned}
$$

and

$$
\begin{bmatrix} \tilde{V}_3^{(1)} \\ \tilde{V}_4^{(1)} \end{bmatrix} = \begin{bmatrix} R^{(1)} V_3^{(1)} \\ R^{(1)} V_4^{(1)} \end{bmatrix}. \qquad (68)
$$

Substituting Eq. (61), (63) and (66) into (60), we obtain the final result for the matrix-matrix product

$$
Y = AV \approx Y^{(0)} + P^{(0)} \left(\tilde{Y}^{(1)} + P^{(1)} \tilde{Y}^{(2)} \right) \qquad (69)
$$

where

$$
Y^{(0)} = \begin{bmatrix} A_{11}^{(0)} V_1^{(0)} + A_{12}^{(0)} V_2^{(0)} \\ A_{21}^{(0)} V_1^{(0)} + A_{22}^{(0)} V_2^{(0)} \\ \vdots \\ A_{87}^{(0)} V_7^{(0)} + A_{88}^{(0)} V_8^{(0)} \end{bmatrix} \qquad (70)
$$

$$
\tilde{Y}^{(1)} = \begin{bmatrix} \tilde{Y}_1^{(1)} \\ \tilde{Y}_2^{(1)} \\ \tilde{Y}_3^{(1)} \\ \tilde{Y}_4^{(1)} \end{bmatrix} = \begin{bmatrix} \tilde{A}_{12}^{(1)}\tilde{V}_2^{(1)} \\ \tilde{A}_{21}^{(1)}\tilde{V}_1^{(1)} \\ \tilde{A}_{34}^{(1)}\tilde{V}_4^{(1)} \\ \tilde{A}_{43}^{(1)}\tilde{V}_3^{(1)} \end{bmatrix} \qquad (71)
$$

$$
\tilde{Y}^{(2)} = \begin{bmatrix} \tilde{Y}_1^{(2)} \\ \tilde{Y}_2^{(2)} \end{bmatrix} = \begin{bmatrix} \tilde{A}_{12}^{(2)}\tilde{V}_2^{(2)} \\ \tilde{A}_{21}^{(2)}\tilde{V}_1^{(2)} \end{bmatrix} \qquad (72)
$$

To summarize, matrix-matrix product computation includes the following steps:

1. Compute $\tilde{V}^{(1)}$ in Eq. (65) and (68), and compute $\tilde{V}^{(2)}$ in Eq. (62);
2. Compute $Y^{(0)}$ in Eq. (70), $\tilde{Y}^{(1)}$ in Eq. (71) and $\tilde{Y}^{(2)}$ in Eq. (72);
3. Interpolate and cumulative sum in Eq. (69);

Note that all operations in step-2 are dense matrix-matrix product, well suited for dense linear algebra libraries optimized for GPU and TPU. The total number of super- and sub-diagonal blocks is upper bounded by twice the number of super- and sub-diagonal blocks at level-0, which is $2N_b^{(0)}$. The run time of each dense matrix-matrix product is $O(N_r^2 d)$. So the total run time is $5N_b^{(0)} N_r^2 d = 5LN_r d = O(Ld)$. Here we leave out N_r since it is a constant model hyper-parameter.

The coarsening in step-1 and interpolation in step-3 all use sparse matrices with fixed sparsity patterns. Hence matrices $P^{(l)}$ and $R^{(l)}$ are never explicitly formed and applying them can be easily done with standard library functions. Take Jax Numpy library as an example, coarsening can be done with sum() along row axis and interpolation can be done with repeat() along row axis. For this reason, step-1 and step-3 only have dense matrix operations as well.

The formulation of the matrix-matrix product for the general level-M case is

$$
\begin{aligned}
Y &= AV = Y^{(0)} + P^{(0)}(\tilde{Y}^{(1)} + P^{(1)}(\tilde{Y}^{(2)} \\
&+ P^{(2)}(\cdots + P^{(M-2)}\tilde{Y}^{(M-1)})\cdots)). \qquad (73)
\end{aligned}
$$

This formulation is a direct consequence of the nested attention matrix structure and can be derived similarly as Eq. (69).