Transformers: A Graph Processing Perspective

Manish Sri Sai Surya Routhu*, Sai Dheeraj Yanduru*, Nathaniel Tomczak, Sanmukh Kuppannagari
Department of Computer and Data Sciences, Case Western Reserve University
Contact: {mxr809, sxy874, nkt8, sxk1942}@case.edu

***Abstract*—Transformers, a variant of AI/ML models that utilize the attention mechanism to capture interactions in sequential data, have significantly advanced a variety of scientific and engineering applications. However, attention implementations suffer from high computational and memory complexity requirements and despite efforts to reduce their complexity, the context length over which they can capture interactions is still several orders of magnitude lower than desired. The fundamental problem is that even though these models capture pairwise interactions between data elements, they are viewed as sequence of tensor operations on tensor data, thereby, limiting avenues of optimization.**

In this work, we take the first steps towards re-imagining transformer based models as graph computing pipelines by implementing the attention mechanism. Our expectation is that this view will not only dramatically increase scalability by unlocking parallelism along the context length dimension, but will also enable researchers to apply the vast library of graph analytics algorithms to obtain better insights into the inner working of these models. We implement graph algorithms for attention mechanism and conduct extensive experimentation by varying sequence length, token dimensions, and sparsity factor and observer near linear reduction in computation time with sparsity factor.

***Index Terms*—Transformer Models; Attention Mechanism; Graph Computing; PyTorch**

I. Introduction

Transformer models utilize a mechanism known as attention in order to represent interactions within data and have been utilized in an array of applications including large language modeling [1], molecular design [2], cancer pathology classification [3], and genomic sequence modeling [4]. These models derive their power from an operation called *attention mechanism*. Given a sequence of tokens of length L, attention mechanism extracts pairwise similarities between the tokens. This information drives the training of Transformer models. The sequence length [4] and the model size [5] are the two key determinants of the representation power of these models.

With the existing three dimensional parallelism in training pipelines [6], [7] — data, model/pipeline, and tensor parallelism, enabling significant increases in the model size, sequence length is increasingly becoming the limiting factor in increasing the representation power of these models. For applications such as genomics, at least 4-5 orders of magnitude of increase in sequence length is needed [4]. Attention mechanism is the key bottleneck and remains so despite the numerous techniques developed for improving their efficiency (Section II-C).

The key limitation of the existing techniques is that they view Transformer models as a set of tensor operations on tensor data. This prevents efficient parallelization across the sequence length dimension. However, as the role of attention is to capture pairwise interactions between the tokens, we advocate for a graph representation which more naturally captures such interactions.

Specifically, We propose a training pipeline that views the input Tensor data (sequence of tokens) as a graph with tokens as vertices and their interactions (that changes along the layers) as edges and perform computations on the graph as determined by the deep learning model. We expect this view to both increase scalability by unlocking parallelism along the context length dimension and enable researchers to apply the vast resources of graph analytics libraries to obtain better insights into the inner working of these models.

In this work, we take initial steps towards re-imagining transformer based models as graph computing pipelines by implementing the attention mechanism using graph algorithms. The specific contributions of this paper are:

- We present a graph computation perspective for random sparse attention mechanism that is a key attention mechanism employed in popular sparse attention mechanisms such as BigBird, and linformer [8]–[10].
- We develop a vertex-centric algorithm to efficiently implement random sparse attention mechanisms.
- We conduct extensive experiments that vary sequence length, token embedded dimensions, and the sparsity factor. Our results demonstrate near-linear speedup with respect to the sparsity factor demonstrating the potential of graph algorithms to reduce attention execution times.
- We discuss the implications of our proposed graph computing view as well as potential future research directions.

II. Background

A. Transformer Models

Transformer models are encoder-decoder architectures where each transformer layer takes as input a sequence of d dimensional tokens of length L, projects them into a set of d_k dimensional queries packed in a matrix $Q \in R^{L \times d_k}$, a set of d_k dimensional keys packed in a matrix $K \in R^{L \times d_k}$, and a set of d_v dimensional values, packed into a matrix $V \in R^{L \times d_v}$ using learnable weight matrices W_Q, W_K and W_v respectively [11].

Figure 1 demonstrates the computations performed in the attention process. The three matrices from left to right represent the queries, keys, and values corresponding to the L input tokens that are obtained by projection using the learnable weight matrices. The query matrix (leftmost matrix) stores the d_k dimensional queries as rows, the key matrix (middle matrix) stores the d_k dimensional keys as columns, and the value

*Equal Contribution

matrix (rightmost matrix) stores the d_v dimensional values as rows. Mathematically, the attention computation can be represented using Equation 1:

$$Attention(K, Q, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

In practice, a multi-headed attention mechanism is utilized, where the keys, queries, and values are projected to several smaller dimensions, attention is applied in parallel to each projection, and the results are concatenated to obtain the attention output [11].

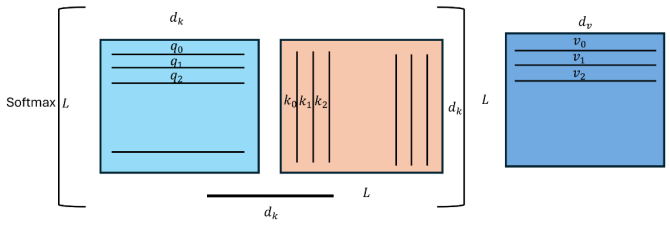

Fig. 1: The core attention mechanism represented as operating on matrices. The cyan represents the queries, the orange the keys (which is also transposed), and the blue the values. d_k represents the size of the embedded dimension.

Intuitive Definition:

Focusing on an individual token i, one can observe that attention performs a soft-lookup. Specifically, the following computations are executed for a token i: we take the query corresponding to the token (row i of the query matrix) and compute a dot-product with each key (columns of the key matrix). A softmax is computed on the L scalar values obtained from the dot products that results in a vector of size L. Due to the softmax operation, the values of the vectors will be between 0-1 and they will sum up to 1. Thus, this vector can be treated as a weight vector corresponding to token i, where weights are obtained by computing similarity between the query corresponding to token i and keys corresponding to all the other (including self) tokens. Finally, each row of the value matrix will be scaled using the corresponding element of the weight vector and added to produce the output corresponding to token i.

$$\begin{aligned} O_i &= \sum_{j=1}^{L} w_{ij} V_j \\ w_{ij} &= sim(Q_i, K_j) \end{aligned} \quad (2)$$

This can be summarized as: for a token i, it outputs the weighted sum of values, where weights are normalized similarity scores between the keys corresponding to the values and the query corresponding to the token as shown in Equation 2. Here O_i is the output of the attention mechanism for token i, V_j is the value vector corresponding to token j obtained by projection using W_v, K_j is the key vector corresponding to token j obtained by projection using W_k and Q_i is the query vector corresponding to token i obtained by projection using W_q.

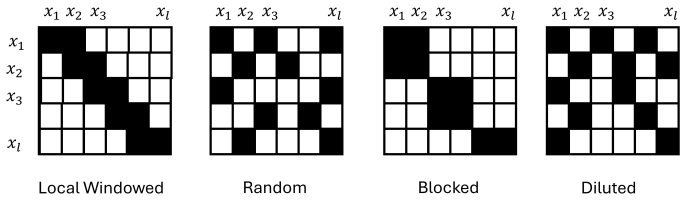

Fig. 2: Various examples of sparse attention mechanisms that alter the ability for certain tokens to interact with one another.

The key takeaway here is that attention captures interactions between pairs of input tokens and can be represented as a *graph operation*, especially, if not every pair needs to interact with each other.

B. Complexity Analysis

Assuming a sequence length of L for keys, queries, and values, the computational complexity of the attention mechanism as per Equation 1 is quadratic in L for the two matrix multiplications. The memory requirement to store the temporary output of QK^T is $L \times L$. This makes sequence length a bottleneck in these models.

C. Attention Mechanism Algorithms

Extensive research has been conducted on reducing the computational and memory requirements of the attention mechanism to improve the length of context that they can capture. We categorize and summarize them below.

Sparse Attention: Instead of computing L^2 dot products between each query-key pair (rows of matrix Q and columns of matrix K), research in this category focused on reducing the number of key-query pairs that interact with each other. Key-query pair interactions are represented using a binary adjacency matrix of size $L \times L$ known as *attention mask*. Several attention mask patterns have been proposed including global attention [8], [9], [12], local windowed attention [8], [9], [13], random attention [8]–[10], blocked attention [14], [15], and context window dependent diluted attention [16] as shown in Figure 2.

The sparsity of these masks is directly related to the *Sparsity Factor*, S, which functions as a multiplicative constant for the number of dot product computations: $O((1 - S) * L^2)$. This paper describes the *Sparsity Factor* as the level of sparsity of the attention mask and can be calculated by the following equation:

$$S = \frac{NNZ}{TE} \quad (3)$$

where NNZ is the number non-zero elements in the mask and TE is the total number of elements in the mask. As an example, a sparsity factor of 0 means a fully-dense attention matrix while a sparsity factor of 0.75 means that 75% of the

attention matrix is masked. As the sparsity factor approaches 1, meaning the mask is extremely sparse, the number of dot product computations will trend towards order $O(L)$.

Dense Attention: Works in this category have focused on reducing the memory consumption of attention mechanisms while keeping the computational complexity unchanged. Flash attention uses a matrix tiling technique to compute partial attentions [17], [18]. Tile sizes are determined based on the available memory. Memory-efficient attention computes matrix multiplications as successive dot products to ensure that the partial results fit into the available memory [19]. Other works such as [20], [21] perform low-rank approximations of matrices to reduce computation complexity while performing all-to-all attention.

Kernel Approximation: Works in this category leverage the idea of kernel methods in machine learning models to avoid the explicit computation of the $L \times L$ weight matrix (whose entries are w_{ij} from Equation 2). Specifically, they use a learnable kernel $\mathcal{K}(q_i, k_j)$ to directly output the scalar weight value w_{ij}. Development of kernels that can obtain high accuracy with lower computational complexity is the focus of these works [22]–[24].

Coarsening Attention: For scientific domains such as genomics [4] and protein analysis [25] that require capturing long range interactions, techniques have been developed to capture interactions between aggregation of tokens as opposed to pairs of individual tokens. These works trade-off the range of interaction with granularity to obtain the best possible performance. Techniques such as convolution [4], [26], Fourier transform [4], [27], fixed k-mer or frequency-based byte-pair encodings [28], and downsampling [29] are typically used to aggregate tokens.

D. Attention Mechanism Implementation in PyTorch

In pytorch, `scaled_dot_product_attention()` implements the attention mechanism [30]. Sparse attention is achieved by passing the attention mask as one of the arguments to the function. However, despite the presence of a sparse attention mask, the pytorch implementation performs a dense matrix multiplication between $Z = QK^T$. For all 0 values in the attention mask, it then sets the corresponding values in the Z matrix as $-\infty$. Then, the softmax operation is performed which results in a sparse matrix that is multiplied by the V matrix to obtain the final output. Due to the requirement of dense matrix multiplication, it is expected that sparsity will not significantly reduce the computations.

III. Graph Processing View of Attention Mechanism

In this work, our focus is mainly on the sparse attention techniques discussed in Section II-C. While dense attention and kernel approximation techniques can be trivially modeled using our proposed graph processing view, as they rely on L^2 all-to-all interactions between the tokens, further exploration is needed to determine if a graph processing perspective results in any benefits or not. This will be a topic for future exploration. We do not focus on coarsening attention in this work.

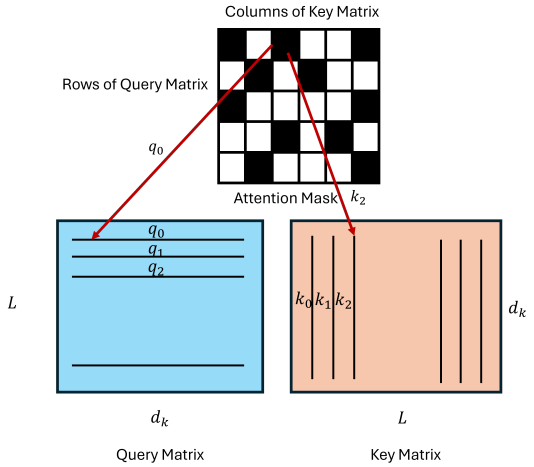

Fig. 3: The product of query and key matrices in the presence of an attention mask. Solid entries of attention denote an interaction whereas white entries denote no interaction between the corresponding query-key pairs.

A. Attention as Graph Computations

We revisit the operations of attention under sparse attention mask in Figures 3 and 4. Figure 3 denotes the operation QK^T in Equation 1. In the presence of an attention mask — typically a 1-0 mask where 1 at position i, j represents an interaction between query i and key j and 0 denotes no interaction — dot products are performed between the d_k dimensional query and key vectors only when they interact as per the mask. The product produces an $L \times L$ matrix as shown in Figure 4, where the solid entries are the results of dot-products (and will interact with value entries), whereas the transparent entries (shaded in blue) are invalid entries. These invalid entries are set to $-\infty$ and then a row-wise softmax is computed on this matrix. For each row, softmax takes the vector of size L as input and produces a probability vector (all values between 0-1 and all values sum up to 1). The invalid entries receive a value of 0 after softmax computation. One can observe that the output pattern of QK^T as well as the softmax is an $L \times L$ matrix which resembles the attention mask. This output matrix is multiplied by the value matrix. The final output will be a weighted sum of the values with weights stored in rows of the matrix that was output after the softmax operation.

Our graph computing view for a transformer layer, as well as the sequence of operations described above, proceeds as follows: for an input d dimensional token sequence $X_1, X_2, \ldots, X_L$, we build a graph G with vertex set $V = XV_1, XV_2, \ldots, XV_L$ with attributes as the corresponding d dimensional tokens and no edges. The projection matrices W_Q, W_K, W_V are applied to each vertex separately to produce three vector attributes for each vertex.

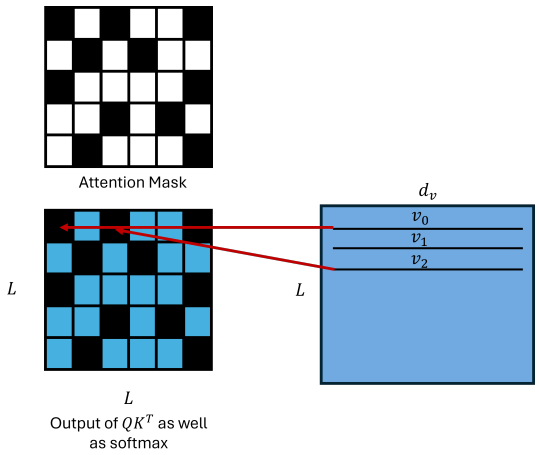

Fig. 4: The bottom left matrix denotes a masked output of QK^T and its softmax operation

Now, when we need to compute the attention mechanism, an edge set E is overlaid onto the graph. The edge set represents the attention mask and may vary for different layers. In this work, we solely focus on graph algorithms for attention mechanism, creating the entire training and inference pipelines using a graph computing perspective will be carried out in future. In the next section, we describe our proposed vertex-centric graph algorithms for the sequence of operations executed in attention mechanisms.

B. Vertex-Centric Graph Algorithms for Attention

We are given a graph $G = (V, E)$ with a sequence of L tokens. The attributes of vertex $XV_i \in V$ are denoted as Q_i, K_i, V_i corresponding to the projection of token X_i with W_Q, W_K, W_V matrices, respectively. The attention mask can be used to generate the edge set E. We provide vertex-centric algorithms for the three operations that need to be performed on this graph to implement the attention mechanism. In addition to the attributes, we assume that additional local memory of size L is available for each vertex to store the output softmax weights. We denote this as *weight memory*.

QK^T: For each vertex XV_i, a dot product is performed the attribute Q_i and the attribute K_j of each neighbor $j \in \mathcal{N}(XV_i)$, where $\mathcal{N}(XV_i)$ denotes the neighbor set of XV_i. The neighbor attributes need to be communicated using the *pull* operation as specified in Algorithm 1. The output of the dot products are stored into the local weight memory. Algorithm 1 denotes the graph algorithm to implement this step.

Softmax: For each vertex XV_i, simply compute softmax using the weights stored in the local weight memory. The output values are stored in-place at the same location. This step does not require any communication between the vertices. We assume that for a vertex XV_i, the weights are stored as $W_i = w_{ij} \forall j \in \mathcal{N}(XV_i)$.

Algorithm 1: Vertex Centric Algorithm for QK^T

Input: Graph G = (V, E). Attributes:
$Q_i, K_i \; \forall XV_i \in V$
Output: Weights $W_i \; \forall XV_i \in V$

foreach $XV_i \in V$ **do** in parallel
  foreach $j \in \mathcal{N}(XV_i)$ **do**
    $K \leftarrow Pull(K_j)$;
    $W_{ij} \leftarrow Q_i.K$;
  $W_i \leftarrow W_{ij} \forall j \in \mathcal{N}(XV_i)$

Product with V: For each vertex XV_i, initialize an output buffer of size d_v with values set to 0. Now for each neighbor j, pull the attribute V_j, scale it using w_{ij} and add it to the output buffer. Algorithm 2 specifies the graph algorithm.

Algorithm 2: Vertex Centric Algorithm for product with V

Input: Graph G = (V, E). Attributes:
$K_i, W_i \; \forall XV_i \in V$.
Output: Output tokens $O_i \; \forall XV_i \in V$

$O_i \leftarrow 0$;
foreach $XV_i \in V$ **do** in parallel
  foreach $j \in \mathcal{N}(XV_i)$ **do**
    $V \leftarrow Pull(V_j)$;
    $O_i \leftarrow O_i + W_{ij}.V$;

IV. Methods

The graph processing algorithm described above was created without a focus on the pre-attention projection layer, meaning that input tensors were treated as though they had already been projected into the embedding space. Only single-headed attention was performed. Additionally, the CUDA block size used for all graph processing tests was 16 threads. All experiments conducted within this paper utilized the Case Western Reserve University High Performance Computing (HPC) resource with a node configured with a NVIDIA Tesla V100 32GB GPU (SXM2).

For correctness, the algorithm written in CUDA was tested in tandem with an identical implementation of scaled dot product attention to PyTorch's. The same Q, K, V, and $mask$ tensors (in CSR form) were utilized between the two implementations with a float32 data type. Output tensors from each operation were compared to one another with an absolute tolerance of 10^{-4} over a series of sequence lengths, embedded dimensions, and sparsity factors. All of the output tensors compared between the two implementations were found to match, verifying that the graph processing method correctly calculates attention.

A series of experiments that measured the execution speed of the graph processing attention implementation were conducted to observe time benefits of the algorithm. Three different parameters were varied during testing:

- **Sequence Length:** 32, 64, 128, 256, 512, and 1024.
- **Embedded Dimension:** 32, 64, 128, 256, and 512.
- **Sparsity Factor:** 0.03, 0.2, 0.4, 0.6, and 0.8.
  - *Sparsity was random, future research will explore alternative masking methods and their affects on performance.*

Each combination of parameters had 10 warm up runs that were not timed and 15 subsequent runs that were recorded in milliseconds. The same Q, K, and V tensors were utilized during these 25 runs, while the $mask$ tensor was randomly generated each time.

For evaluation, the combinations were grouped by sequence length. A scale factor equivalent to the mean time of most sparse and middle-sized (embedded dimension of 128) was used to scale all times for those members in the like-sequence length set. Therefore, the times are no longer absolute, rather they are relative to one another and should be used to observe trends and not to pull numerical results. In Figure 5, each subgraph contains a dashed black line at $y = 1$ which intersects the pink line (embedded dimension 128) at a sparsity factor of 0.8, indicating that its mean was the scale factor. For each combination, the 15 recorded and scaled times were averaged and the standard deviation was calculated.

V. Experimental Evaluations

A. Results

Figure 5 showcases subgraphs containing all combinations of variables from Section IV tied to a specific value of sequence length. A subgraph demonstrates trends of average scaled runtimes (as described in Section IV). The bars protruding from the top and bottom of points are the scaled time standard deviations. The dashed black line at $y = 1$ indicates that the scale factor is equal to the mean runtime from the middle-sized embedded dimension for a specific group of sequence length.

Observing Figure 5 it is apparent that there is a general trend where for all lines of equal embedded dimension, as the sparsity factor increases (attention mask becomes more sparse), the mean runtime decreases. With greater sparsity, there are fewer non-zero elements meaning that fewer calculations must be conducted in order to perform the attention operation.

However, this trend is not absolute. In the subgraph for sequence length of 128, with an embedded dimension of 512 the scaled mean runtime between a sparsity factor of 0.2 and 0.4 is roughly constant (with a slight increase). And, in the subgraph for a sequence length of 256, with an embedded dimension of 512 the scaled mean runtime between sparsity factors of 0.03 and 0.2 shows a noticeable increase. Lastly, in the subgraph for a sequence length of 1024, the line for an embedded dimension of 64 between a sparsity factor of 0.03 and 0.2 is relatively constant. These deviances from the trend could be due to nonoptimal code choices for the specific hardware that was used; the CUDA kernels were not optimized nor were the block sizes tuned. Additionally, these non-linearities are more apparent with larger tensors (greater sequence length and embedded dimension). In future work, a greater number of tests will have to be conducted across multiple pieces of hardware (using NVIDIA's Nsight program for more precise profiling) in order to get a better perspective of these trends.

Within each sequence length subgraph, the larger embedded dimensions (bigger tensor) have steeper decreasing trends. This would be expected as for each increase in the sparsity factor, a larger tensor would have a greater decrease in the number of operations it must perform.

VI. Discussion on Implications

Techniques to achieve sequence parallelism across multiple GPUs in existing deep learning pipelines [7], [31] are co-designed and tightly coupled with their attention masks, thereby, severely limiting their ability to scale up arbitrary models. Moreover, they all rely on all-gather operations which maybe suboptimal in sparse attention mechanisms.

In contrast, a graph computation view, such as the one developed in this work, will enable us to develop sophisticated distributed algorithms and graph partitioning schemes potentially leading to infinite scaling along the sequence dimension.

This work, where we used a naive single GPU based vertex centric algorithm to demonstrate the potential of graph algorithms to reduce the computation time of these models, is the first work that takes a graph processing perspective of Transformer models. We expect this perspective to significantly increase the scalability and representation power of these models.

Acknowledgements

This work made use of the High Performance Computing Resource in the Core Facility for Advanced Research Computing at Case Western Reserve University.

VII. Conclusion

Sequence length is an important variable for transformers as it directly increases both the context length for tokens and information bandwidth for the model. Sparse attention methods can help greatly increase sequence lengths of transformers without the need for accelerators with greater quantities of memory. This paper has proved that graph processing methods can output the identical results to those calculated by PyTorch's scaled dot product attention implementation. Additionally, a general trend is observed showing that graph processing methods decrease the mean runtime required for attention as the attention mask grows increasingly sparse.

References

[1] C.-Y. Hsieh, C.-L. Li, C.-K. Yeh, H. Nakhost, Y. Fujii, A. Ratner, R. Krishna, C.-Y. Lee, and T. Pfister, "Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes," 2023. [Online]. Available: https://arxiv.org/abs/2305.02301

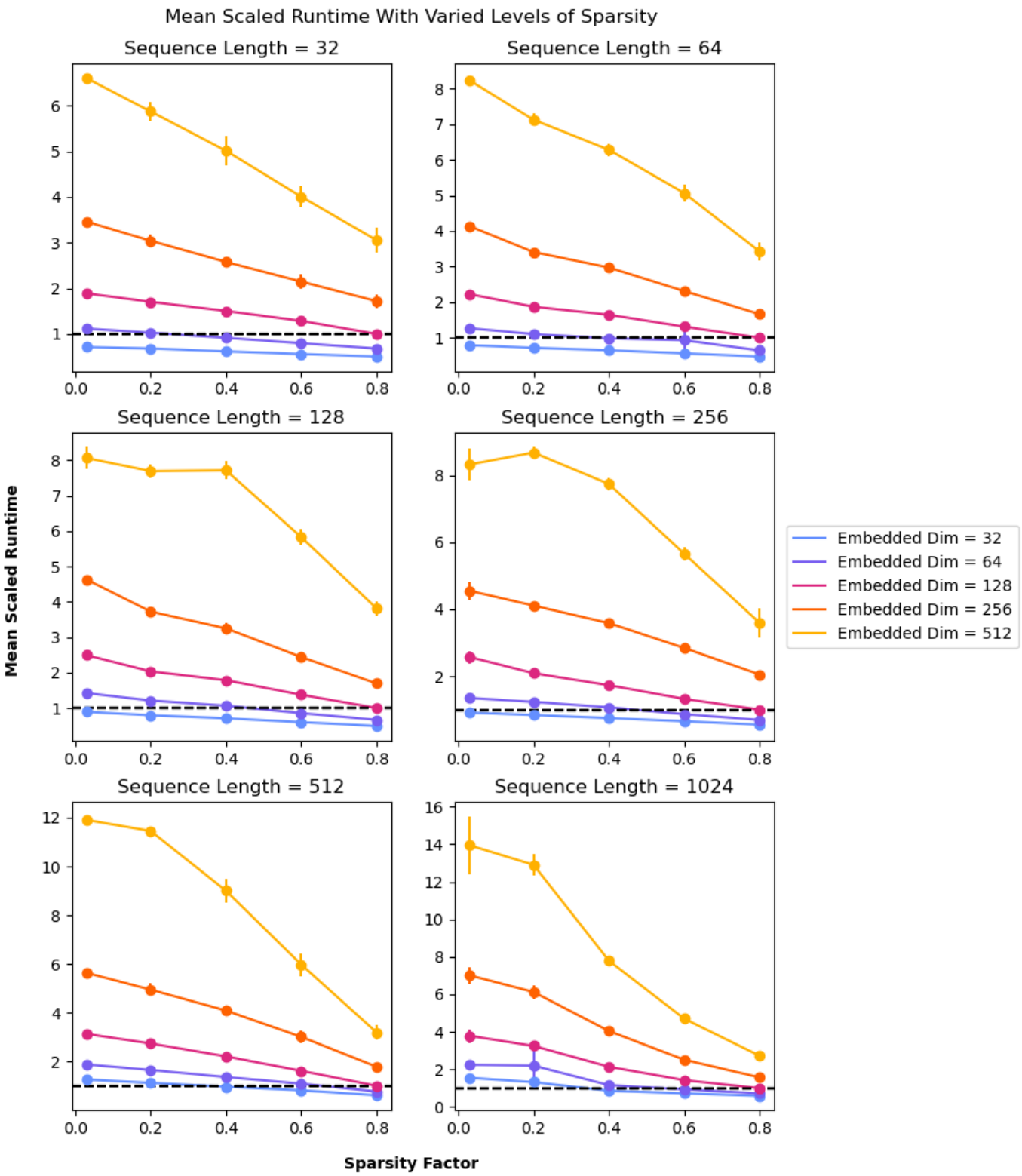

Fig. 5: Graphs showcasing the trends of scaled mean runtimes (described in Section IV) for groups of identical sequence length. Each color of line indicates a different sized embedded dimension used during the tests.

[2] D. Bhowmik, P. Zhang, Z. Fox, S. Irle, and J. Gounley, "Enhancing molecular design efficiency: Uniting language models and generative networks with genetic algorithms," *Patterns*, vol. 5, no. 4, 2024.

[3] M. Chandrashekar, I. Lyngaas, H. A. Hanson, S. Gao, X.-C. Wu, and J. Gounley, "Path-bigbird: An ai-driven transformer approach to classification of cancer pathology reports," *JCO Clinical Cancer Informatics*, vol. 8, p. e2300148, 2024.

[4] E. Nguyen, M. Poli, M. Faizi, A. Thomas, M. Wornow, C. Birch-Sykes, S. Massaroli, A. Patel, C. Rabideau, Y. Bengio *et al.*, "Hyenadna: Long-range genomic sequence modeling at single nucleotide resolution," *Advances in neural information processing systems*, vol. 36, 2024.

[5] D. Narayanan, M. Shoeybi, J. Casper, P. LeGresley, M. Patwary, V. Korthikanti, D. Vainbrand, P. Kashinkunti, J. Bernauer, B. Catanzaro, A. Phanishayee, and M. Zaharia, "Efficient large-scale language model training on GPU clusters," *CoRR*, vol. abs/2104.04473, 2021. [Online]. Available: https://arxiv.org/abs/2104.04473

[6] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3505–3506.

[7] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[8] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[9] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang *et al.*, "Big bird: Transformers for longer sequences," *Advances in neural information processing systems*, vol. 33, pp. 17 283–17 297, 2020.

[10] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," *arXiv preprint arXiv:2001.04451*, 2020.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[12] Z. Wu, P. Jain, M. Wright, A. Mirhoseini, J. E. Gonzalez, and I. Stoica, "Representing long-range context for graph neural networks with global attention," *Advances in Neural Information Processing Systems*, vol. 34, pp. 13 266–13 279, 2021.

[13] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[14] H. Liu, M. Zaharia, and P. Abbeel, "Ring attention with blockwise transformers for near-infinite context," *arXiv preprint arXiv:2310.01889*, 2023.

[15] H. Liu and P. Abbeel, "Blockwise parallel transformers for large context models," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[16] J. Ding, S. Ma, L. Dong, X. Zhang, S. Huang, W. Wang, N. Zheng, and F. Wei, "Longnet: Scaling transformers to 1,000,000,000 tokens," *arXiv preprint arXiv:2307.02486*, 2023.

[17] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in Neural Information Processing Systems*, vol. 35, pp. 16 344–16 359, 2022.

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

[19] M. N. Rabe and C. Staats, "Self-attention does not need $o(n^2)$ memory," *arXiv preprint arXiv:2112.05682*, 2021.

[20] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *arXiv preprint arXiv:2006.04768*, 2020.

[21] G. I. Winata, S. Cahyawijaya, Z. Lin, Z. Liu, and P. Fung, "Lightweight and efficient end-to-end speech recognition using low-rank transformer," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 6144–6148.

[22] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are rnns: Fast autoregressive transformers with linear attention," in *International conference on machine learning*. PMLR, 2020, pp. 5156–5165.

[23] Z. Qin, X. Han, W. Sun, D. Li, L. Kong, N. Barnes, and Y. Zhong, "The devil in linear transformer," *arXiv preprint arXiv:2210.10340*, 2022.

[24] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser *et al.*, "Rethinking attention with performers," *arXiv preprint arXiv:2009.14794*, 2020.

[25] R. Chowdhury, N. Bouatta, S. Biswas, C. Floristean, A. Kharkar, K. Roy, C. Rochereau, G. Ahdritz, J. Zhang, G. M. Church *et al.*, "Single-sequence protein structure prediction using a language model and deep learning," *Nature Biotechnology*, vol. 40, no. 11, pp. 1617–1623, 2022.

[26] Z. Wu, Z. Liu, J. Lin, Y. Lin, and S. Han, "Lite transformer with long-short range attention," *arXiv preprint arXiv:2004.11886*, 2020.

[27] J. Lee-Thorp, J. Ainslie, I. Eckstein, and S. Ontanon, "Fnet: Mixing tokens with fourier transforms," *arXiv preprint arXiv:2105.03824*, 2021.

[28] Y. Ji, Z. Zhou, H. Liu, and R. V. Davuluri, "Dnabert: pre-trained bidirectional encoder representations from transformers model for dna-language in genome," *Bioinformatics*, vol. 37, no. 15, pp. 2112–2120, 2021.

[29] Ž. Avsec, V. Agarwal, D. Visentin, J. R. Ledsam, A. Grabska-Barwinska, K. R. Taylor, Y. Assael, J. Jumper, P. Kohli, and D. R. Kelley, "Effective gene expression prediction from sequence by integrating long-range interactions," *Nature methods*, vol. 18, no. 10, pp. 1196–1203, 2021.

[30] "Scaled Dot Product Attention," https://shorturl.at/cek3U.

[31] S. A. Jacobs, M. Tanaka, C. Zhang, M. Zhang, L. Song, S. Rajbhandari, and Y. He, "Deepspeed ulysses: System optimizations for enabling training of extreme long sequence transformer models," *arXiv preprint arXiv:2309.14509*, 2023.