Sequoia: Hierarchical Self-Attention Layer with Sparse Updates for Point Clouds and Long Sequences

Hugo Sonnery *
School of Computer Science
McGill University / Mila
Montréal, QC (Canada)
hugo.sonnery@mail.mcgill.ca

Thuan Anh Trang *
FPT Software
Ho Chi Minh City (Vietnam)
thuantna@fsoft.com.vn

Thieu N. Vo
Ton Duc Thang University / FPT Software
Ho Chi Minh City (Vietnam)
vongocthieu@tdtu.edu.vn

Siamak Ravanbakhsh
School of Computer Science
McGill University / Mila
Montréal, QC (Canada)
siamak@cs.mcgill.ca

Truong Son Hy
Halicioglu Data Science Institute
University of California San Diego
San Diego, USA
tshy@ucsd.edu

Abstract

Lack of efficiency of self-attention in dealing with long inputs has motivated many efficient transformer variants. We consider hierarchy as an interesting route toward sparsity. This can be achieved by forming a tree on top of the input tokens and limiting the span of attention, based on the branching factor, thereby reducing the complexity of the self-attention layer to $O(n \log(n))$. This paper provides preliminary results in creating such transformer architecture, called `Sequoia`[1] The proposed method leverages the hierarchy within a single self-attention layer and sequentially applies the attention mechanism to the tree in a bottom-up fashion. Although the proposed architecture makes minimal design choices, we provide ablation studies to support them. Our preliminary results on point cloud tasks and sequence classification benchmarks suggest favourable performance compared to state-of-the-art transformer architectures without resorting to efficient CUDA implementation. The code for the project is available at `https://github.com/Fsoft-AIC/Sequoia`

1 Introduction

Transformers [30] surpassed the existing SOTA on almost all input modalities through the combined feat of parallel training and dot-product attention. From natural language processing to graph-based video understanding, transformers have displayed competitive performance on standard benchmarks over the past few years. However, their quadratic complexity in the sequence length $\Omega(n^2)$ makes them intractable for long sequences [28]. Therefore, a considerable body of literature on *efficient*

[1] `Sequoia`. ***Sequ**ential **A**ttention*

transformers is dedicated to scaling the vanilla transformer beyond the 512 tokens frontier with multiple approaches to resolving their scaling/performance tradeoff.

Drawing on the intuition that self-attention iteratively refines tokens' embeddings using all-to-all communication, some efficient transformers sparsify the attention matrix to prioritize the computation of the most useful interactions. In this work, we assume a hierarchical underlying generative process and encode this hierarchy in the transformer architecture. This choice translates to introducing "virtual nodes" corresponding to internal nodes of the tree. Each node only attends to its children, parents and siblings in the tree, reducing the computational complexity of the attention layer to quasi-linear in the size of the input. In the following, after a brief review of the related works, we elaborate on this hierarchical attention scheme and present preliminary results on point cloud classification and segmentation, as well as sequence classification.

2 Related work

Most efficient transformers either simplify the computation of the attention matrix or get rid of it altogether. Reformer [11] is a content-based sparse attention transformer that matches tokens together into buckets using locality-sensitive hashing before applying local attention. Others employ learnable sparsity patterns, like Sinkhorn transformers [26] which induce sparsity by sorting the keys and values matrices such that local heuristics can be applied in the computation of the scaled-dot-product. Most efficient transformers papers such as [9], [1], [23] and [41] resort to similar sparsity heuristics in order to get rid of the bulk of the computations in the dot-product attention in a structured way, allowing for efficient implementation without custom CUDA kernels. They often rely on local information processing, only selecting a few tokens as "global tokens" which will attend to every token and be attended to by all tokens. Further away from sparsity-based approaches, Transformer-XL [6] is a memory-based transformer that adds recurrent connections across segments by reusing the hidden states of the model while processing tokens of the previous segment. The work closest to ours is BP Transformer [40] which uses binary partitioning of a sequence in order to integrate information from long-term context in a fine-to-coarse fashion. Our model relies on sequential local message-passing updates in a tree, as opposed to BP Transformer which achieves global information sharing by computing intermediate nodes' embeddings in parallel directly from their leaf nodes instead of their direct children. Also, it relies on sequential ordering of the tokens to construct the dynamic hierarchy, which our model does not.

3 Method

Our proposed attention scheme consists in sequentially applying an attention-based update at various levels of a k-ary tree constructed on top of the input tokens. The leaves of the tree correspond to actual tokens of the input, while intermediate nodes are hubs for information propagation at different resolutions, thus allowing global information sharing[2].

The construction of this tree depends on the geometry of the input data itself; for point cloud data, the tree structure can use the Euclidean distance for deciding the neighbourhoods; for example, one could use KD-tree, hierarchical clustering, or hierarchical K-nearest neighbours to construct the tree. For sequential data, the timestamp of tokens can be used to define neighbourhoods, where higher nodes in the tree represent increasingly larger intervals.

We consider the most relevant neighbours to update a token's embedding to be its *children*, *siblings* and *ancestors*. Global information sharing is achieved by sequentially updating the nodes' embeddings in a bottom-up fashion. More concretely, $\mathcal{A}^{\text{children}}(i)$ refines node i's embedding by computing scalar dot-product attention between its query Q_i and its children' keys K_j.

$$\mathcal{A}^{\text{children}}(i) = \text{Attention}\left[Q_i, (K_j, V_j)_{j \in \text{children}(i)}\right] \tag{1}$$

A similar process is used to produce an embedding based on ancestors $\mathcal{A}^{\text{ancestors}}(i)$, and siblings $\mathcal{A}^{\text{siblings}}(i)$. One would then need to combine these embeddings, e.g., using an MLP or even another

[2]We refer the interested reader to the appendix for a thorough presentation of the mathematical formalism of *efficient transformer models* and a detailed explanation of our model's attention mechanism.

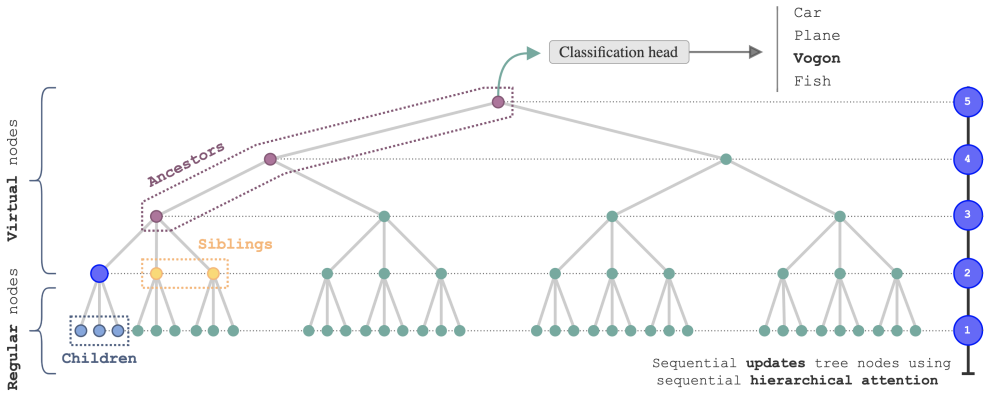

Figure 1: Sequoia, our proposed sequential hierarchical attention scheme.

attention mechanism to update the embedding for node i. For simplicity, in our experiments we use average pooling.

To further simplify the model, one could limit the attention span to only include the children and parents of each node. Sibling and ancestor information could, in principle, propagate through the parent. However, in practice, we found the effect of siblings' attention to be significant in performance. Note that this choice does not affect the asymptotic complexity of the proposed layer; the complexity in both run-time and memory remains quasi-linear in the input size $\mathcal{O}(n \log(n))$. In practice, the added cost remains negligible given the tree only has a few layers, and the number of nodes drops significantly in each layer.

4 Experimental results

4.1 Point clouds

To demonstrate the efficiency of Sequoia on point clouds, we conduct experiments on two tasks:

- **Shape classification:** ModelNet40 [36] dataset is a classification dataset that contains 12,311 3D models categorised into 40 classes. In this experiment, we use 9,843 samples for training and 2,468 for validation and testing. To measure the performance of each model, we report the testing overall accuracy and average class accuracy in Table 1.

- **Part segmentation:** ShapeNetPart [17] dataset is a part segmentation dataset, which contains 16,881 synthetic point clouds from 16 classes. Specifically, 14,006 samples are used for training and 2,874 for validation and testing. In this data, each object has 2 to 6 parts, resulting in 50 parts in total. To evaluate the model, we use class average IoU and instance average IoU, which is reported in Table 1.

In each task, we compare the performance of our model with the normal soft-max attention and other models specifically designed for point clouds. Furthermore, we also run an ablation study to find the most optimal configuration for our model and compare its complexity with other approaches. The results of the ablation study are reported in C.2.1 and settings of all experiments are described in C.3.1.

Although we do not produce the SOTA results on the two tasks for point cloud, Sequoia is a lightweight attention-based model with competitive results. Furthermore, with the new algorithm for tree construction, we can avoid $O(n^2)$ complexity when calculating distance for sampling and finding nearest neighbours in other point-based methods. We significantly outperform softmax attention while requiring much lower complexity in both computation and memory for training and inference.

Method	Accuracy	mAcc
VoxNet [16]	85.9	83.0
MVCNN [25]	90.1	–
PointNet [19]	89.2	86.0
DGCNN [34]	92.9	90.2
GridGCN [38]	93.1	91.3
Set Transformer [13]	90.4	–
PointNet++ [20]	91.9	88.4
PointConv [35]	92.5	–
KPConv [29]	92.9	–
PointTransformer [42]	93.7	90.6
Softmax attention	89.5	87.4
Sequoia	**92.0**	**88.4**

Method	Inst IoU	Class IoU
PointNet [19]	71.9	43.7
DGCNN [34]	85.1	82.3
PointNet++ [20]	85.1	81.9
PointConv [35]	85.7	82.6
PointCNN [14]	86.1	84.6
PointTransformer [42]	86.6	83.7
Softmax attention	78.8	74.9
Sequoia	**82.7**	**79.5**

Table 1: Shape Classification results on ModelNet40 (left table) and Part Segmentation results on ShapeNetPart (right table)

.

4.2 Sequence classification

Long Range Arena [27] is a sequence classification benchmark specifically designed for assessing transformers' capabilities to efficiently process and summarize long sequences [3]. In order to accurately measure the quality of the inductive bias of sequential hierarchical attention, we train our model on an array of complementary classification tasks from LRA : IMDB (text), Cifar10 (image), Listops (mathematical sequence parsing), Retrieval (byte-level document matching) and Pathfinder (image). In the experimental results below 2, we report for each model the classification accuracy over a total of 50,000 training steps on 4 RTX8000 GPUs :

Method	IMDB	Cifar10	Listops	Retrieval	Patfhinder-32
Nystrom-64	62.11	56.89	36.64	81.55	75.62
Linformer-64	56.03	43.57	38.66	76.22	90.22
Performer-64	62.46	38.59	18.4	78.62	69.90
Linear	50.98	22.28	17.79	49.41	50.36
Softmax attention	60.87	48.45	39.62	OOM	88.61
None attention	60.50	37.03	37.1	80.42	50.36
Sequoia (ours)	**62.72**	**49.88**	**37.70**	**67.04**	**58.44**

Table 2: Sequence Classification results on Long Range Arena benchmark for efficient transformers.

Sequoia proves to be a good general-purpose model for sequence classification on a wide range of input sequence modalities. Sequoia outperforms the Vanilla baseline (Softmax attention) on IMDB and Listops, which demonstrates that sparsifying the attention matrix using sequential attention can not only lower the computational complexity of the model, but also provide a beneficial inductive bias. Although our model is theoretically efficient with $O(n\log(n))$ complexity, it only becomes more runtime and memory efficient than vanilla attention at $9,000$ tokens. Other efficient transformer baselines typically outperform vanilla attention as soon as 512 tokens. This currently restrict Sequoia's usage for very long-range sequence modeling and classification applications (see figure 7 in the appendix).

Our primary purpose in this paper was to explore how to better learn long-term dependencies, using Sequoia's sparse hierarchical attention as a prototype. Our current implementation involves tensor re-indexing of the keys and values. In order to go beyond a simple proof-of-concept of Sequoia's inductive bias of hierarchical attention, we intend to leverage the sparsity of the

[3]We refer the interested reader to [27] for an exhaustive description of each dataset of the LRA benchmark.

attention graph with specialized tensor operations in future work in order to make Sequoia more computationally efficient.

5 Conclusion

Sequoia is part of a larger research effort to better understand how attention-based models could be efficiently sparsified by means of hierarchy. In this work-in-progress paper, we have introduced two essential ingredients towards such unification : (i) sequential updates of tokens' embeddings on a (ii) hierarchical tree induced by the input tokens. Sequoia demonstrates solid performance on a range of tasks, from point cloud classification and segmentation to sequence classification across multiple input modalities. Besides, it is general-purpose and requires little adaptation to a particular input (as opposed to most point cloud understanding models), as long as a hierarchical k-ary tree can be built over the set of input tokens.

Unlike most efficient transformer models, `Sequoia` is readily applicable to the causal (autoregressive) generation mode. We plan to investigate its application to language generation in future work. Among other directions that we would like to explore in the future are generalizing the tree structure to a lattice, thereby increasing the number of pathways between input tokens, and; adaptive neighbourhood sizes that dynamically learn to identify the underlying hierarchy during the training. In future work, we intend to decrease the computational cost of our model by using hardware-optimized implementations of sparse scalar dot product.

Acknowledgements

We would like to thank anonymous reviewers for their input. This project was in part supported by FPT Software AI Center and the CIFAR AI chairs program. Computational resources are provided by FPT, Mila and the Digital Research Alliance of Canada.

References

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

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[3] S. Chaudhari, V. Mithal, G. Polatkan, and R. Ramanath. An attentive survey of attention models. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 12(5):1–32, 2021.

[4] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, D. Luan, and I. Sutskever. Generative pretraining from pixels. In *International conference on machine learning*, pages 1691–1703. PMLR, 2020.

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

[6] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[7] M. Dehghani, S. Gouws, O. Vinyals, J. Uszkoreit, and Ł. Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

[8] M.-H. Guo, J.-X. Cai, Z.-N. Liu, T.-J. Mu, R. R. Martin, and S.-M. Hu. Pct: Point cloud transformer. *Computational Visual Media*, 7(2):187–199, Apr 2021. ISSN 2096-0662. doi: 10.1007/s41095-021-0229-5. URL `http://dx.doi.org/10.1007/s41095-021-0229-5`.

[9] Q. Guo, X. Qiu, P. Liu, Y. Shao, X. Xue, and Z. Zhang. Star-transformer. *arXiv preprint arXiv:1902.09113*, 2019.

[10] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

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

[12] C. Krzysztof, L. Valerii, D. David, S. Xingyou, G. Andreea, S. Tamas, H. Peter, D. Jared, M. Afroz, K. Lukasz, et al. Rethinking attention with performers. *Proceedings of ICLR*, 2021.

[13] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR, 2019.

[14] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen. Pointcnn: Convolution on x-transformed points. *Advances in neural information processing systems*, 31, 2018.

[15] X. Ma, X. Kong, S. Wang, C. Zhou, J. May, H. Ma, and L. Zettlemoyer. Luna: Linear unified nested attention. *Advances in Neural Information Processing Systems*, 34:2441–2453, 2021.

[16] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 922–928. IEEE, 2015.

[17] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas, and H. Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 909–918, 2019.

[18] C. Park, Y. Jeong, M. Cho, and J. Park. Fast Point Transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[19] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[20] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017.

[21] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[22] J. W. Rae, A. Potapenko, S. M. Jayakumar, and T. P. Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

[23] A. Ravula, C. Alberti, J. Ainslie, L. Yang, P. M. Pham, Q. Wang, S. Ontanon, S. K. Sanghai, V. Cvicek, and Z. Fisher. Etc: Encoding long and structured inputs in transformers. In *2020 Conference on Empirical Methods in Natural Language Processing (EMNLP 2020)*, 2020. URL `https://www.aclweb.org/anthology/2020.emnlp-main.19.pdf`.

[24] A. Roy, M. Saffar, A. Vaswani, and D. Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021.

[25] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.

[26] Y. Tay, D. Bahri, L. Yang, D. Metzler, and D.-C. Juan. Sparse sinkhorn attention. In *International Conference on Machine Learning*, pages 9438–9447. PMLR, 2020.

[27] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

[28] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, 2022.

[29] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6411–6420, 2019.

[30] 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*, 30, 2017.

[31] A. Vyas, A. Katharopoulos, and F. Fleuret. Fast transformers with clustered attention. *Advances in Neural Information Processing Systems*, 33:21665–21674, 2020.

[32] N. Wang, G. Gan, P. Zhang, S. Zhang, J. Wei, Q. Liu, and X. Jiang. Clusterformer: Neural clustering attention for efficient and effective transformer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2390–2402, 2022.

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

[34] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

[35] W. Wu, Z. Qi, and L. Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 9621–9630, 2019.

[36] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[37] Y. Xiong, Z. Zeng, R. Chakraborty, M. Tan, G. Fung, Y. Li, and V. Singh. Nyströmformer: A nyström-based algorithm for approximating self-attention. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 14138–14148, 2021.

[38] Q. Xu, X. Sun, C.-Y. Wu, P. Wang, and U. Neumann. Grid-gcn for fast and scalable point cloud learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5661–5670, 2020.

[39] Z. Ye, Q. Guo, Q. Gan, X. Qiu, and Z. Zhang. Bp-transformer: Modelling long-range context via binary partitioning. *arXiv preprint arXiv:1911.04070*, 2019.

[40] Z. Ye, Q. Guo, Q. Gan, X. Qiu, and Z. Zhang. Bp-transformer: Modelling long-range context via binary partitioning, 2019. URL `https://arxiv.org/abs/1911.04070`.

[41] 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*, 33:17283–17297, 2020.

[42] H. Zhao, L. Jiang, J. Jia, P. H. Torr, and V. Koltun. Point transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 16259–16268, 2021.

A Background

A.1 Transformers

Transformer block Let L be the number of layers, H the number of attention heads, N the maximum sequence length and d the token feature dimension at the output of the transformer block [4]. A Transformer is defined by the iterative composition of *transformer blocks* sharing the following form :

$$\mathcal{T}(x) = \mathcal{F}(\mathcal{A}(x) + x) \tag{2}$$

Scaled dot-product self-attention Scaled-dot-product attention updates embeddings through a weighted pooling of tokenwise features pooled across the entire input sequence.
Let us first compute (`query`, `key`, `value`) pairs for each token :

$$Q, K, V = XW_Q, XW_K, XW_V \tag{3}$$

with $W_Q \in \mathbb{R}^{d \times d_k}$,$W_K \in \mathbb{R}^{d \times d_k}$ and $W_V \in \mathbb{R}^{d \times d_v}$.

Token i's embedding is updated using the following formula:

$$X'_i = \mathcal{A}_{W_Q, W_K, W_V}(X, i) = \sum_{j \in [1,N]} \mathcal{S}(Q_i, K_j) V_j \tag{4}$$

In scaled-dot-product attention, the interaction scores $\mathcal{S}(\cdot, \cdot)$ are computed as follows:

$$\mathcal{S}(Q_i, K_j) = \frac{\exp(Q_i K_j^T)}{|\mathcal{S}_i|} \times \frac{1}{\sqrt{d_k}} \tag{5}$$

with $|\mathcal{S}_i|$ the normalization factor associated with token i:

$$|\mathcal{S}_i| = \sum_{j \in [1,N]} \exp(Q_i K_j^T) \tag{6}$$

Multiheaded attention Multiheaded *self-attention* parallelizes the computation of attention into various subspaces of the latent space. We first split each input into H different subspaces of latent dimension $d_k^h = \frac{d_k}{H}$ for queries and keys and $d_v^h = \frac{d_v}{H}$:

$$\mathcal{A}_{\text{multiheaded}}(X) = [X'^1 :: \cdots :: X'^h :: \cdots :: X'^H] W_O \tag{7}$$

where X'^h is the output of the self-attention mechanism for head $h \in [1, H]$:

$$\mathcal{A}_{\text{singlehead}}(X) = \mathcal{A}_{W_Q^h, W_K^h, W_V^h}(X) \tag{8}$$

with $W_Q^h \in \mathbb{R}^{d \times d_k^h}$,$W_K^h \in \mathbb{R}^{d \times d_k^h}$, $W_V^h \in \mathbb{R}^{d \times d_v^h}$ and $W_O \in \mathbb{R}^{d_v \times d}$.

A.2 Transformers for point clouds

Regarding point clouds, transformer only drew considerable attention from researchers in the recent years. For example, Point Cloud Transformer [8] proposes a method that directly applies full attention between all points in a point cloud which results in reasonable performance but has limitation in terms of scalability. Thereafter Point Transformer [42] tackles this issue by imposing hierarchical structure from PointNet++[20] which calculates attention only for a group of points in a local region. Fast Point Transformer[18] further optimizes the function to boost the speed of self-attention based model to 129 times faster than Point Transformer [42] in semantic segmentation.

[4] For the sake of simplicity, we omit the layer and head indices l and h, the extension to the multi-layer / multi-head case being straightforward.

B Our model

B.1 Proposed model

In order to learn a principled hierarchy of distributed representations, we construct a recursive k-ary tree over the input sequence of tokens, which correspond to its leaf nodes. The other intermediate levels of the tree up until the root node will henceforth be denoted *virtual nodes*. Let us now introduce some preliminary notation :

- $X^0 \in \mathbb{R}^{n \times d}$ is the embedding matrix of the tokens in the input sequence.
- $\tau = \tau(X)$ is the recursive k-ary tree built using **branching factor** k [5].
- $n^\tau = \sum_{\lambda=0}^{\Lambda-1} n_\lambda^\tau$ is the total **number of nodes in the tree** (including both leaf nodes and virtual nodes).
- Λ^τ is the total **number of levels in the tree** [6].
- $X_0^\tau \in \mathbb{R}^{n^\tau \times d}$ is the embedding matrix of the nodes in the augmented input sequence.
- X_λ^τ corresponds to the embeddings matrix of the nodes of level λ.
- $I_\lambda, |I_\lambda| = n_\lambda^\tau$ is the list of the indices of the nodes in a given layer λ of τ.

`Sequoia`'s **hierarchical message-passing attention** scheme consists in taking into account three different *neighbourhoods* based on the tree structure before updating a node $i \in I_\lambda$'s embedding :

1. **Children** $C_{\lambda,i}^\tau$ (all nodes $j \in I_{\lambda-1} \mid Parent[j] = i$).
2. **Siblings** $S_{\lambda,i}^\tau$ (all nodes $j \in I_\lambda \mid Parent[j] = Parent[i]$).
3. **Ancestors** $A_{\lambda,i}^\tau$: (all nodes $j \in I_\lambda \cup \cdots \cup \in I_{\Lambda-1} \mid \underbrace{Parent \circ \cdots \circ Parent}_{\Lambda - \lambda \text{ times}}[i] = j$).

Our proposed attention scheme consists in applying :

$$\mathcal{A}^{\text{ent}}(\cdot) = \phi\left[\mathcal{A}^{\text{children}}(\cdot); \mathcal{A}^{\text{siblings}}(\cdot); \mathcal{A}^{\text{ancestors}}(\cdot)\right] \tag{9}$$

with ϕ an appropriate fusion layer mixing the updated embeddings of each node according to the three different attention types (*children*, *siblings* and *ancestors*) on the nodewise axis.

For each node $i \in I_\lambda$, we compute its **interaction scores** with its **children**, **siblings** and **ancestors**, and apply **softmax normalization** on the resulting attention weights (scaled dot-products) across each of the three neighbourhoods. We then **pool the values** from its neighbours weighted by the corresponding attention scores. More formally, let Q_i be the query embedding associated with node i, and $(K_j, V_j)_{j \in \mathcal{V}}$ be the key / value embeddings associated with nodes j in the neighbourhood $\mathcal{V}$.

$$\mathcal{A}^{\text{children}}(X_\lambda^\tau, i) = Attention_{\,W_Q^c, W_K^c, W_V^c}\left[Q_i, (K_j, V_j)_{j \,\in\, \text{children}(i \mid \tau, \lambda)}\right] \tag{10}$$

$$\mathcal{A}^{\text{siblings}}(X_\lambda^\tau, i) = Attention_{\,W_Q^s, W_K^s, W_V^s}\left[Q_i, (K_j, V_j)_{j \,\in\, \text{siblings}(i \mid \tau, \lambda)}\right] \tag{11}$$

$$\mathcal{A}^{\text{ancestors}}(X_\lambda^\tau, i) = Attention_{\,W_Q^a, W_K^a, W_V^a}\left[Q_i, (K_j, V_j)_{j \,\in\, \text{ancestors}(i \mid \tau, \lambda)}\right] \tag{12}$$

We finally fuse the updated embeddings proposed for node i by each of the three attention types using simple average pooling.

$$\phi_{\text{average}}[C_{\lambda,i}^\tau, S_{\lambda,i}^\tau, A_{\lambda,i}^\tau] = \frac{C_{\lambda,i}^\tau + S_{\lambda,i}^\tau + A_{\lambda,i}^\tau}{3} \tag{13}$$

[5] Extension to the more general setting where the branching factor is not the same across all levels, but is instead specified separately for each level in the tree $(k_0, \cdots, k_{\Lambda-1})$ is straightforward.

[6] To make notations more readable, we now drop the $\cdot^\tau$ superscript.

Sequential updates Instead of updating all nodes (both true and virtual) simultaneously (in *parallel*), we first update the values of the results in little added computational complexity while increasing the expresivity of our model. Indeed, since all sequential updates are applied in a single transformer layer, the root node has global receptive field at the end of the first layer. Technically, if we use bottom-up sequential updates, we need two layers in order to achieve global information sharing between any pair of points. Multiple update schemes (priority orderings of the message passing schemes) can be considered, such as bottom-up -> top-down, which we leave for future work.

$$\forall \lambda \in [0, \Lambda - 1], \forall i \in I_\lambda,\ X^\tau_{\lambda,i} := \mathcal{A}^{\text{HMP}}(X^\tau, \lambda, i) \tag{14}$$

Please note that this scheme dramatically reduces the computational complexity of the transformer model, especially in the deeper levels of the tree.

C Experimental details

C.1 Computational complexity analysis

The total number of nodes (both true and virtual) in the tree is :

$$\begin{aligned} |I| = \sum_{\lambda=0}^{\Lambda-1} |I_\lambda| = \sum_{\lambda=0}^{\Lambda-1} \lceil n \times \frac{1}{k^\lambda} \rceil &= n \times \left[\frac{1 - \left(\frac{1}{k}\right)^\Lambda}{1 - \frac{1}{k}} \right] + \mathcal{O}(\Lambda) \\ &= n \left[\frac{k^\Lambda - 1}{k^{\Lambda-1}(k-1)} \right] + \mathcal{O}(\Lambda) \approx (\leq)\ n \times \underbrace{\left[\frac{k}{k-1} \right]}_{\Delta_k} \end{aligned} \tag{15}$$

Thus, $|I| \leq n \times \Delta_k$, with $\Delta_k = \frac{k}{k-1} \geq 1$ being the **dilation factor** of the tree. Let us now examine the computational complexity of `Sequoia`'s efficient attention scheme, both in terms of *runtime* and *memory* :

$$\mathcal{C}_{\texttt{Sequoia}}(n) = \mathcal{C}_{\text{children}}(n) + \mathcal{C}_{\text{siblings}}(n) + \mathcal{C}_{\text{ancestors}}(n) \tag{16}$$

$$\mathcal{C}_{\text{children}}(n)\ ,\ \mathcal{C}_{\text{siblings}}(n)\ ,\ \mathcal{C}_{\text{ancestors}}(n) = \sum_{\lambda=1}^{\Lambda-1} \mathcal{C}_{\text{children}}(n, \lambda)\ ,\ \sum_{\lambda=0}^{\Lambda-1} \mathcal{C}_{\text{siblings}}(n, \lambda)\ ,\ \sum_{\lambda=0}^{\Lambda-2} \mathcal{C}_{\text{ancestors}}(n, \lambda) \tag{17}$$

$$\begin{aligned} \forall \lambda \in [0, \Lambda - 1]\ \mathcal{C}_{\text{children}}(n, \lambda), \mathcal{C}_{\text{siblings}}(n, \lambda) &= \mathcal{O}(|I_\lambda| \times k) \\ \forall \lambda \in [0, \Lambda - 1]\ \mathcal{C}_{\text{ancestors}}(n, \lambda) &= \mathcal{O}(|I_\lambda| \times (\Lambda - \lambda)) = \mathcal{O}(|I_\lambda| \times \Lambda) \end{aligned} \tag{18}$$

Using the upper bound on the number of nodes in the tree constructed by `Sequoia`, we get :

$$\begin{aligned} \mathcal{C}_{\text{children}}(n) &= \mathcal{O}\left(\sum_{\lambda=0}^{\Lambda-1} |I_\lambda| \times k \right) \approx (\leq)\ n \times k \times \Delta_k \\ \mathcal{C}_{\text{ancestors}}(n) &= \mathcal{O}\left(\sum_{\lambda=0}^{\Lambda-1} |I_\lambda| \times \Lambda \right) \approx (\leq)\ n \times \Lambda \times \Delta_k \end{aligned} \tag{19}$$

For $\mathcal{C}_{\text{siblings}}(n)$, we obtain a similar bound as for $\mathcal{C}_{\text{children}}(n)$. We finally derive the following class of *quasilinear* computational complexity for our proposed attention model :

$$\mathcal{C}_{\texttt{Sequoia}}(n) = \mathcal{O}\left(n \times \left[2 \times \left[\frac{k}{k-1} \right] + \Lambda \right] \right) = \mathcal{O}(n \times (k + \log(n)) \tag{20}$$

Indeed, the tree depth is such that $\Lambda = \lceil \frac{\log(n)}{\log(k)} \rceil \leq \frac{\log(n)}{\log(k)} + 1 = \mathcal{O}(\log(n))$. In particular, we notice that this upper bound on $\mathcal{C}_{\texttt{Sequoia}}(n)$ does not depend on the mode of computation of attention inside a given transformer block (`bottom-up` / `top-down` or `parallel`), and that the total complexity of `Sequoia`'s attention scheme is dominated by the attention computed at the first level of the tree[7]. A more thorough investigation of the scaling capabilities of our model for various **branching factors** k demonstrates two interesting phenomena :

[7] Precisely, *siblings* attention on the first level and, to a lesser extent, ancestors attention on the first level.

- For **small** k (in our experimental setting, typically for $\mathbf{k \leq 8}$), the total complexity is dominated by the sequential nature of computations, even though each level of attention computation is relatively inexpensive.
- For **large** k ($\mathbf{k \geq 32}$), the receptive field of the first level's sibling attention dramatically increases, approximating vanilla attention.

Therefore, in order to strike a good balance between quality of approximation and computational efficiency, we chose to set $k = 16$ for the remainder of this paper unless indicated otherwise.

C.2 Experiment settings

C.2.1 Experimental settings on point cloud

Construct tree from point cloud dataset: Regarding the setup of the experiments, as the point clouds are unordered, it is non-trivial to build the tree. Here, for each point cloud, we divide it into boxes iteratively and the center of the box is the parent nodes of the points inside. The points are considered as the leaf of the tree. Notably, the feature of the parent nodes is aggregated by pooling features of its children. Furthermore, the nodes at higher level of the tree have the larger dimension in terms of features since their contains information for larger region in a point cloud, compared to the lower nodes.

Adapt normal softmax attention to point cloud dataset: In terms of the baseline using normal softmax attention, directly applying the methods to the point cloud results in excessive computational requirements. Therefore, in this network, we only allow the point to pay attention to its K nearest neighbors.

Experiment hyper-parameters and optimizer: In our experiments, we find that the tree with a depth of 3 and a branching factor equal to 3 is the most effective setup because the larger tree increases complexity without improving the performance. In terms of attention type, we set our attention as bottom-up and allow a node to pay attention to its direct-parent, children, and siblings. Moreover, the weight of all Transformer layers is shared, so the number of parameters remains the same when we stack the layers in the networks. In all experiments, our model has 4 Transformer layers. The other configurations are kept the same as other works. In particular, the optimizer used in both experiments is SGD (learning rate = 0.05, momentum = 0.9, and weight decay = 0.0001) and we train the model for 200 epochs with a batch size of 32 for ModelNet40[36] and 16 for ShapeNetPart[17].

C.2.2 Experimental settings on LRA

We train our model (Sequoia) on a representative array of sequence classification tasks taken from the LRA suite :

1. **IMDB** : text (byte-level) classification.
2. **Cifar10** : image classification from raw digits read in raster order.
3. **Listops** : parsing of hierarchical mathematical structures.
4. **Retrieval** : byte-level document matching.
5. **Pathfinder-32-Baseline** : image classification for path connectedness prediction.

For experimental simplicity, we decided to use a standard model backbone configuration throughout all sequence classification experiments. More specifically, our model has 4 transformer layers, with an intermediate feature dimension of 512 split across 4 heads. We use dropout of $p = 0.1$ on the attention matrix. We also share the same optimizer settings for all datasets : batch size of 32 with a learning rate of 0.001 with linear warmup. The accuracy reported corresponds to the test accuracy obtained on the best performing model according to validation rounds carried out every 2,000 training steps throughout the training procedure. For Pathfinder-32, we use the "baseline" version of the task.

Whenever we refer to efficient transformer models by `nystrom-64`, `linformer-64` or `performer-64`, 64 respectively corresponds to the number of landmarks of Nystromformer, Linformer's k parameter and the RP dimension of the Performer model. Regarding our proposed model, Sequoia, we use $k = 16$ and a bottom-up sequential attention scheme (i.e. we start by

updating the leaf nodes up until the root node). As an input to the MLP classification head, we provide the root node of the tree. Furthermore, for our sequence classification task, we get rid of ancestors attention (i.e. there is only bottom-up information propagation between the nodes across the tree). Indeed, we do not need our model to make tokenwise predictions but only an overall prediction for the entire sequence, and this allows us to further reduce the computational burden of our model. We leave the study of the respective contribution to final performance of the three nodewise attention neighbourhoods (children, siblings and ancestors), as well as ablations regarding the key ingredients to the success of our model, to future work.

C.3 Efficiency benchmarking

C.3.1 Efficiency Benchmarking on Point Cloud:

In this section, we conduct experiments on the number of points, the number of sibling neighbors, the number of Transformer layers, and the node feature to observe their impacts on the performance, complexity, and runtime of the model. All the results are reported in 3, 4, 5, and 6 respectively.

As observed from 3 and 4, the Sequoia with 4 layers and 8 siblings' attention gives us the best result. Increasing those hyper-parameters may negatively affect both performance and computational requirements. In terms of the feature of node 5, directly using the root node for classification achieves better accuracy than pooling inner nodes and leaf nodes. This result is reasonable because the root node has the highest number of feature dimensions which can store more meaningful global features of the point cloud. Eventually, the benchmark in table 6 shows how our model scale with the number of points in the point cloud.

Numbers of siblings	Overall Accuracy	Training time	Infer time
4	89.3	16.0ms	9.2ms
8	**92.0**	16.8ms	9.6ms
16	90.6	18ms	10ms

Table 3: Shape Classification results on the ModelNet40 dataset with different siblings neighbours

Layers	Overall Accuracy	Training time	Infer time	FLOPs	Parameters
1	88.4	3.6ms	2.8ms	0.23G FLOPs	10.9M
2	88.5	7.2ms	5.3ms	0.46G FLOPs	10.9M
4	**92.0**	16.8ms	9.6ms	0.95G FLOPs	10.9M
5	90.7	18.0ms	10.2ms	1.15G FLOPs	10.9M

Table 4: Shape Classification results on the ModelNet40 dataset with different layers

Type	Overall Accuracy
Root Node	**92.0**
Average Inner Nodes	90.5
Average Leaf Nodes	90.2

Table 5: Shape Classification results on the ModelNet40 dataset with different types of nodes

Numbers of points	FLOPs	Infer time
64	0.35G	3.4ms
256	0.51G	3.9ms
1024	0.95G	9.6ms
4096	2.62G	24.8ms

Table 6: Shape Classification with different number or points

C.4 Efficiency benchmarking for long-range sequence classification

Computing scalar-dot-product between queries and the sparse selection of keys (children, siblings and ancestors) via tensor re-indexing is highly hardware inefficient given the sparse nature of Sequoia's computational graph compared to full vanilla attention. This could be done more efficiently using `scatter` functions which we leave to future work. We also intend to merge the three attention types (computed one after the other) into a single multiresolution attention function (computed in parallel) which will further help improve Sequoia's efficiency.

Sequence length	**Sequoia** (ours)	**Softmax**
2,048	1.65s	0.91s
4,096	3.35s	1.71s
8,192	6.89s	6.85s
16,384	14.13s	26.33s
32,768	28.93s	`OOM`
65,536	57.39s	`OOM`

Table 7: Efficiency benchmarking of Sequoia for multiple sequence lengths (10-samples batch on 1 RTX8000 GPU). We observe that Sequoia becomes more runtime-efficient than Vanilla attention at around 9,000 tokens.