Zero-Shot Conversion to Monarch-Structured Attention

Can Yaras [1] **Alec S. Xu** [1] **Pierre Abillama** [1] **Changwoo Lee** [1] **Laura Balzano** [1]

Abstract

We propose `MonarchAttention` – a novel approach to sub-quadratic attention approximation via Monarch matrices, an expressive class of structured matrices. Based on the variational form of softmax, we describe an efficient optimization-based algorithm to compute an approximate projection of softmax attention onto the class of Monarch matrices with $\Theta(N\sqrt{N}d)$ computational complexity and $\Theta(Nd)$ memory/IO complexity. Unlike previous approaches, `MonarchAttention` is both (1) transferable, yielding minimal performance loss with no additional training, even when replacing every attention layer of the transformer, and (2) hardware-efficient, utilizing the highest-throughput tensor core units on modern GPUs. With optimized kernels, `MonarchAttention` achieves substantial speed-ups in wall-time over `FlashAttention-2`: $1.4\times$ for shorter sequences ($N = 256$), $4.5\times$ for medium-length sequences ($N = 4K$), and $8.2\times$ for longer sequences ($N = 16K$). We demonstrate the quality of `MonarchAttention` on diverse tasks and architectures in vision and language problems, showing that it flexibly and accurately approximates softmax attention in a variety of contexts.

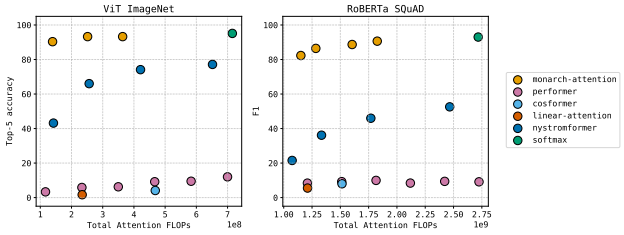


Figure 1: **Zero-shot conversion of attention layers for image classification and question answering.** We vary hyperparameters for various baselines to evaluate model quality vs compute tradeoff. *Left.* Top-5 accuracy vs. total attention FLOPs across all layers for ViT on ImageNet. *Right.* F1 score vs total attention FLOPs across all layers for RoBERTa on SQuAD.

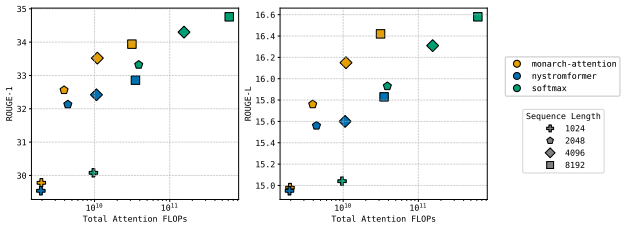


Figure 2: **Zero-shot conversion of attention layers for long sequence summarization.** We vary the sequence length of the text to be summarized to evaluate model quality vs compute tradeoff. We report recall-based ROUGE-1 and ROUGE-L scores vs. total attention FLOPs across all layers for BART on BookSum-chapters.

1. Introduction

Over the past decade, transformers (Vaswani et al., 2017) have become the dominant architecture for generating and processing various data modalities, such as text (Brown et al., 2020), images (Dosovitskiy et al., 2021), and speech (Radford et al., 2023). Central to the transformer's success is *attention*, the mechanism through which complex interactions within sequential data are captured through weighted combinations of embeddings at every position in the sequence. Famously, the attention mechanism has a quadratic-time complexity $\Theta(N^2d)$ in the length of the sequence N, where d is the head dimension, which is a key bottleneck for both training and inference, particularly in long sequence problems. To address this, numerous works have proposed sub-quadratic substitutes for attention. Yet, such approaches either (1) are not transferable, requiring training from scratch or fine-tuning of existing models, or (2) do not yield speed-ups in practice (except on extremely long sequences) due to a gap between theoretical complexity and practical considerations for modern GPUs, especially compared to highly optimized implementations (Dao et al., 2022b).

In this work, we propose `MonarchAttention`: a novel

[1]Department of EECS, University of Michigan, Ann Arbor, U.S.A.. Correspondence to: Can Yaras <cjyaras@umich.edu>.

Proceedings of the 42^{nd} International Conference on Machine Learning, Vancouver, Canada. PMLR 267, 2025. Copyright 2025 by the author(s).

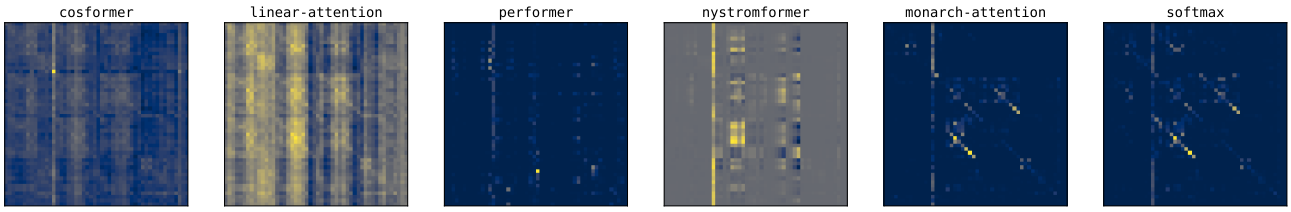


Figure 3: **Approximation of softmax attention via `MonarchAttention`.** By directly optimizing the softmax variational objective constrained to Monarch matrices, `MonarchAttention` yields accurate zero-shot approximation to softmax attention compared to other hardware-friendly, efficient attention baselines. Attention maps extracted from RoBERTa on the SQuAD dataset in Section 4.

sub-quadratic attention substitute based on approximating the attention matrix via *Monarch* matrices (Dao et al., 2022a), a class of expressive structured matrices. We frame the computation of the attention matrix as an optimization problem in terms of the variational form of softmax, and exploit low-dimensional structure in the variational objective when constrained to the set of Monarch matrices – this yields a sub-quadratic $\Theta(N\sqrt{N}d)$-time approximation, where d is the head dimension.

Nearly all approaches to sub-quadratic attention approximate the attention matrix by a structured matrix, specifically low-rank (Wang et al., 2020; Katharopoulos et al., 2020; Choromanski et al., 2021; Qin et al., 2022; Zhang et al., 2024; Xiong et al., 2021), sparse (Child et al., 2019; Chen et al., 2022; Kitaev et al., 2020; Daras et al., 2020), or a combination of the two (Chen et al., 2021; Han et al., 2024) – we review prior work on structured matrices, including Monarch matrices, as well as existing approaches to efficient attention in Appendix A. However, as mentioned before, there are significant drawbacks to these methods in terms of either expressivity or efficiency. `MonarchAttention` achieves the best of both worlds: it is fast and hardware-friendly due to utilization of tensor cores for batched matmuls, while computing highly accurate approximations to the extent that it can directly replace softmax attention with no additional training. We discuss closely related ideas in the literature in Appendix A.

2. Preliminaries

Softmax. The softmax function $\mathbb{R}^N \to \Delta^N$ maps N real numbers to the $(N-1)$-dimensional unit simplex, and is defined as

$$[\mathrm{softmax}(\boldsymbol{z})]_i := \frac{\exp(\boldsymbol{z}_i)}{\sum_j \exp(\boldsymbol{z}_j)}, \ \forall i \in [N]. \tag{1}$$

An alternative definition (Blondel et al., 2019) is given by the following variational form:

$$\mathrm{softmax}(\boldsymbol{z}) := \underset{\boldsymbol{a}\in\Delta^N}{\arg\max} \ \langle \boldsymbol{a}, \boldsymbol{z}\rangle + H(\boldsymbol{a}), \tag{2}$$

where $H(\boldsymbol{a}) = -\sum_i \boldsymbol{a}_i \log \boldsymbol{a}_i$ is Shannon entropy. See Appendix B for equivalence of (1) and (2).

Attention. Given query, key, value matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V} \in \mathbb{R}^{N\times d}$, where N is the sequence length and d is the head dimension, a single head of standard softmax attention[1] computes

$$\boldsymbol{O} = \mathrm{softmax}\left(\boldsymbol{Q}\boldsymbol{K}^\top\right)\boldsymbol{V}, \tag{3}$$

where the softmax function is applied across rows. The computational complexity of attention is $\Theta(N^2 d)$ for each forward pass, because the matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ are data-dependent.

Monarch Matrices. Given $N = m \times b$ for integers m, b, a Monarch matrix $\boldsymbol{M}$ is a permuted block rank-one matrix $\boldsymbol{M} = \boldsymbol{P}^\top \boldsymbol{B}$, where $\boldsymbol{P}$ is a "transpose" permutation and $\boldsymbol{B}$ is a block rank-one matrix

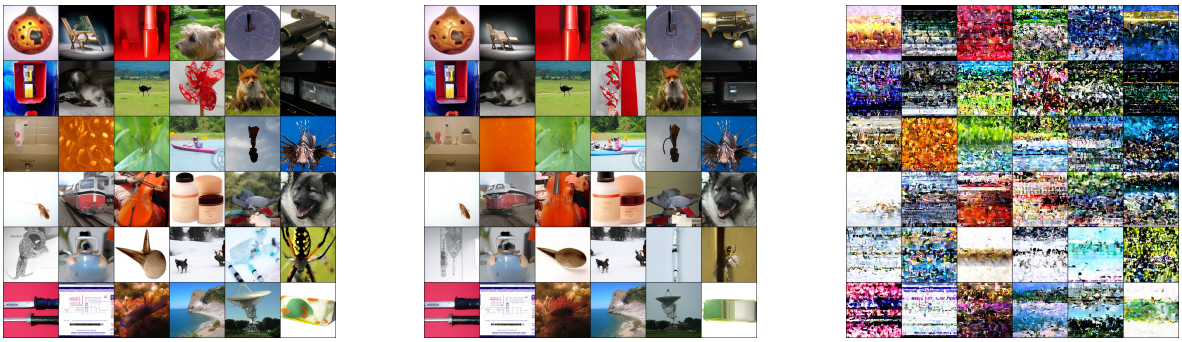
$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_{11} & \dots & \boldsymbol{B}_{1m} \\ \vdots & \ddots & \vdots \\ \boldsymbol{B}_{b1} & \dots & \boldsymbol{B}_{bm} \end{bmatrix}, \tag{4}$$

where $\boldsymbol{B}_{jk} = \boldsymbol{L}_{jk}\boldsymbol{R}_{kj}^\top \in \mathbb{R}^{m\times b}$ for some $\boldsymbol{L}_{jk} \in \mathbb{R}^m, \boldsymbol{R}_{kj} \in \mathbb{R}^b$ for $j \in [b]$ and $k \in [m]$. When $m = b = \sqrt{N}$, storing $\boldsymbol{M}$ requires only $\Theta(N\sqrt{N})$ space, while matrix multiplication (matmul) with a matrix $\boldsymbol{V} \in \mathbb{R}^{N\times d}$ can be computed efficiently in $\Theta(N\sqrt{N}d)$ operations (as opposed to $\Theta(N^2 d)$ for dense matrices) with batched matmuls and transposes. See Appendix C for more details.

3. MonarchAttention

The main goal of `MonarchAttention` is to find a Monarch matrix $\boldsymbol{M} \in \mathbb{R}^{N\times N}$ in $o(N^2 d)$ time such that $\boldsymbol{M} \approx \mathrm{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top)$. Then, we can approximately compute the output $\boldsymbol{O} = \boldsymbol{M}\boldsymbol{V}$ using efficient matmul. We can do this by viewing the softmax operation as an optimization problem via its variational form (2), whose objective can

[1] Typically, the $\boldsymbol{Q}\boldsymbol{K}^\top$ matrix is scaled by a factor of $d^{-1/2}$, but this can be absorbed into $\boldsymbol{Q}$.



(a) Softmax (b) `MonarchAttention` (c) Nyströmformer

Figure 4: **Visual quality of generated images for zero-shot conversion of attention layers.** Example images generated by with softmax (left), `MonarchAttention` (middle), and Nyströmformer (right). Only the first half of the attention layers of DiT are replaced.

be efficiently maximized with exact alternating steps when constrained to Monarch matrices. As shown in Figure 3, this yields highly accurate approximations to the softmax attention matrix.

Softmax Objective. First, from (2) we can write

$$\sigma(\boldsymbol{Q}\boldsymbol{K}^\top) = \underset{\boldsymbol{A}\in\Delta^{N\times N}}{\arg\max} \underbrace{\langle \boldsymbol{A}, \boldsymbol{Q}\boldsymbol{K}^\top\rangle + H(\boldsymbol{A})}_{f(\boldsymbol{A};\boldsymbol{Q},\boldsymbol{K})}, \tag{5}$$

where $\Delta^{N\times N}$ denotes a matrix whose rows lie on Δ^N, and $H(\boldsymbol{A}) = -\sum_{i,j} \boldsymbol{A}_{ij}\log \boldsymbol{A}_{ij}$. For a dense matrix $\boldsymbol{A}$, computing $f(\boldsymbol{A};\boldsymbol{Q},\boldsymbol{K})$ requires $\Theta(N^2 d)$ operations, which is the same as computing $\sigma(\boldsymbol{Q}\boldsymbol{K}^\top)$ directly. However, we are interested in the case where $\boldsymbol{A}$ is a Monarch matrix $\boldsymbol{M} = \boldsymbol{P}^\top\boldsymbol{B}$:

$$\begin{aligned} &f(\boldsymbol{P}^\top\boldsymbol{B};\boldsymbol{Q},\boldsymbol{K}) = \langle \boldsymbol{P}^\top\boldsymbol{B}, \boldsymbol{Q}\boldsymbol{K}^\top\rangle + H(\boldsymbol{P}^\top\boldsymbol{B}) \\ &= \langle \boldsymbol{B}, \widetilde{\boldsymbol{Q}}\boldsymbol{K}^\top\rangle + H(\boldsymbol{B}) = \sum_{j,k} f(\boldsymbol{B}_{jk};\widetilde{\boldsymbol{Q}}_j,\boldsymbol{K}_k), \end{aligned}$$

where $\widetilde{\boldsymbol{Q}} = \boldsymbol{P}\boldsymbol{Q}$, and $\widetilde{\boldsymbol{Q}}_j \in \mathbb{R}^{m\times d}, \boldsymbol{K}_k \in \mathbb{R}^{b\times d}$ are the jth and kth block of rows of $\widetilde{\boldsymbol{Q}}, \boldsymbol{K}$ respectively. Then, for each $j\in[b], k\in[m]$ we evaluate f on the rank-one matrix $\boldsymbol{B}_{jk} = \boldsymbol{L}_{jk}\boldsymbol{R}_{kj}^\top$:

$$\begin{aligned} &f(\boldsymbol{B}_{jk};\widetilde{\boldsymbol{Q}}_j,\boldsymbol{K}_k) \\ =&\langle \boldsymbol{L}_{jk}\boldsymbol{R}_{kj}^\top, \widetilde{\boldsymbol{Q}}_j\boldsymbol{K}_k^\top\rangle - \sum_{l,i} \boldsymbol{L}_{jkl}\boldsymbol{R}_{kji}\log(\boldsymbol{L}_{jkl}\boldsymbol{R}_{kji}) \\ =&\langle \widetilde{\boldsymbol{Q}}_j^\top\boldsymbol{L}_{jk}, \boldsymbol{K}_k^\top\boldsymbol{R}_{kj}\rangle - \left(\boldsymbol{1}^\top\boldsymbol{R}_{kj}\right)\cdot H(\boldsymbol{L}_{jk}) \\ &- \left(\boldsymbol{1}^\top\boldsymbol{L}_{jk}\right)\cdot H(\boldsymbol{R}_{kj}). \end{aligned}$$

Thus, for each $j\in[b], k\in[m]$ we only need $\Theta((m+b)d)$ operations to compute $f(\boldsymbol{B}_{jk};\widetilde{\boldsymbol{Q}}_j,\boldsymbol{K}_k)$ due to $\widetilde{\boldsymbol{Q}}_j^\top\boldsymbol{L}_{jk}$ and $\boldsymbol{K}_k^\top\boldsymbol{R}_{kj}$. We emphasize that the rank-one structure implies separability of the entropy term, meaning we can compute the entropy on $\boldsymbol{L}_{jk}$ and $\boldsymbol{R}_{kj}$ individually and avoid the need to materialize $\boldsymbol{B}_{jk}$, which would incur $\Theta(mb)$ cost as opposed to $\Theta(m+b)$. Since there are $m\cdot b$ many $\boldsymbol{B}_{jk}$ matrices, we have in total $\Theta((m^2b+b^2m)d)$ operations to compute $f(\boldsymbol{M};\boldsymbol{Q},\boldsymbol{K})$, which for $m=b=\sqrt{N}$ is $\Theta(N\sqrt{N}d)$, improving on the dense computation by a factor of $\sqrt{N}$.

Alternating Maximization with Constraints. We will now explain the alternating maximization approach for optimizing f. When $\boldsymbol{L}$ is fixed, the objective is concave in $\boldsymbol{R}$, and vice-versa – therefore, we can derive closed form expressions via KKT conditions for $\boldsymbol{L}$ and $\boldsymbol{R}$ that maximize f with one of $\boldsymbol{L}$ or $\boldsymbol{R}$ fixed, which will constitute a single update step. Evaluating (and therefore differentiating) f w.r.t. $\boldsymbol{L}$ and $\boldsymbol{R}$ can be done in $\Theta(N\sqrt{N}d)$ time, which will be the same complexity as one of these steps. For T steps, this will require $\Theta(TN\sqrt{N}d)$ computation; provided that $T=o(\sqrt{N})$, this will still be sub-quadratic. However, the constraint $\boldsymbol{M}\in\Delta^{N\times N}$ presents a challenge in its current form, since this requires materializing $\boldsymbol{M}$ to check that each entry is non-negative. Instead, we use the fact that $\left(\boldsymbol{L}_{j,:,l}\in\Delta^m,\ \boldsymbol{R}_{kj}\in\Delta^b,\ \forall j\in[b], \forall k,l\in[m]\right) \implies \boldsymbol{M}\in\Delta^{N\times N}$, i.e., slices of $\boldsymbol{L},\boldsymbol{R}$ individually lying on the unit simplex is sufficient to enforce the constraint on $\boldsymbol{M}$. This is easily seen from (6) – obviously if $\boldsymbol{L}_{jkl}, \boldsymbol{R}_{kji}\geq 0$, then $[\boldsymbol{M}_{lk}]_{ji}\geq 0$. Moreover, this also enforces the sum-to-one constraint, as rows of $\boldsymbol{M}$ sum as $\sum_{k,i}[\boldsymbol{M}_{lk}]_{ji} = (\sum_k \boldsymbol{L}_{jkl})(\sum_i \boldsymbol{R}_{kji}) = 1$.

The update steps with full derivation is provided in Appendix D.1. A naïve implementation of the full algorithm is provided in Appendix D.2. We discuss in Appendix D.3 how padding can be incorporated into `MonarchAttention` for when N is not divisible by b. Finally in Appendix D.4 we show how the algorithm is implemented in practice to achieve an optimal $\Theta(Nd)$ IO complexity.

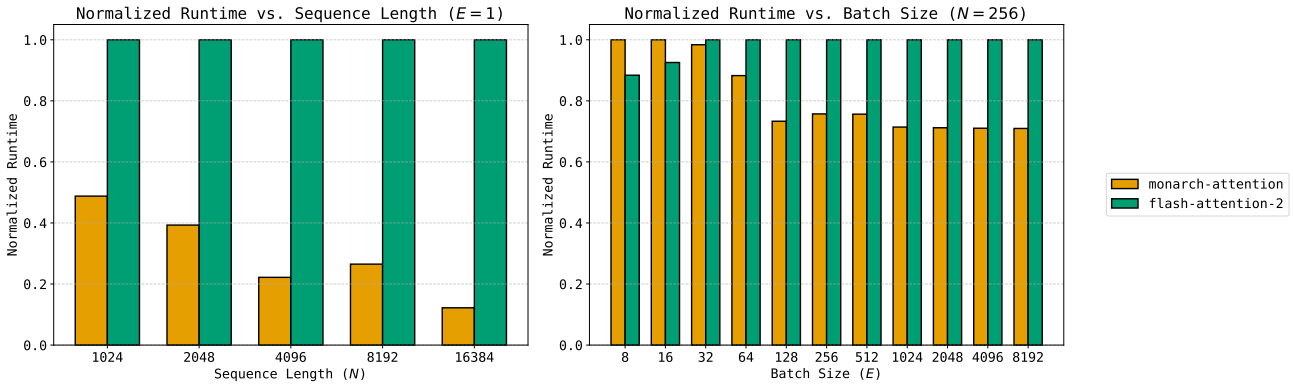


Figure 5: **`MonarchAttention` vs. `FlashAttention-2` run-time across various sequence lengths.** Normalized runtime (1 = slowest, 0 = fastest) of `MonarchAttention` and `FlashAttention-2` on A40 GPU. *Left*: sweep sequence length N with $E = 1$, $H = 12$, and $d = 64$. *Right*: sweep batch size E with $N = 256$, $H = 12$, and $d = 64$.

4. Experiments

In this section, we evaluate the zero-shot performance (no additional training) of `MonarchAttention` for converting pre-trained/fine-tuned transformer attention layers to sub-quadratic attention in four different model/task settings. See Appendix E.1 for more details on the baselines.

Image Classification with Vision Transformer. We convert all attention layers of a trained ViT-B (Dosovitskiy et al., 2021) and evaluate on ImageNet-1K (Russakovsky et al., 2015) for image classification; see Appendix E.2 for more details on the set-up. The results are shown in the left panel of Figure 1. `MonarchAttention` achieves significant improvement over other baselines – compared to the original softmax attention, `MonarchAttention` loses only 5% accuracy to reduce attention FLOPs by 80%, or matches the performance to reduce attention FLOPs by 50%.

Question Answering with Encoder-Only Transformer. We convert a subset of attention layers of a trained RoBERTa-B (Liu et al., 2019) and evaluate on SQuAD1.1 (Rajpurkar et al., 2016) for question answering; see Appendix E.3 for more details on the set-up. The results are shown in the right panel of Figure 1. Once again, `MonarchAttention` achieves significant improvement over other baselines – compared to the original softmax attention, `MonarchAttention` loses only 10 points in F1 score to reduce attention FLOPs by 60%, or matches the performance to reduce attention FLOPs by 35%.

Summarization with Encoder-Decoder Transformer. We convert all attention layers of a trained BART-B (Lewis et al., 2020) encoder and evaluate on BookSum-chapters (Kryściński et al., 2022) for summarization; see Appendix E.4 for more details on the set-up. The results are shown in Figure 2. We see that `MonarchAttention` achieves a strictly better ROUGE score (Lin, 2004) vs. FLOPs tradeoff than even softmax attention, due to accurate and efficient processing of longer sequences. In particular, the $N = 8192$ `MonarchAttention` model improves on the $N = 2048$ softmax attention model by 0.75 on ROUGE-1 and 0.5 on ROUGE-L with slightly fewer FLOPs, while the $N = 8192$ Nyströmformer model with similar FLOPs does strictly worse than softmax.

Image Generation with Diffusion Transformer. We convert a subset of attention layers of a trained DiT-XL (Peebles and Xie, 2023) on ImageNet (Deng et al., 2009) for image generation; see Appendix E.5 for more details on the set-up. Examples of generated images with each method for replacing the first 14 layers are shown in Figure 4, where `MonarchAttention` produces clear images resembling those of softmax attention, while the Nyströmformer ones are extremely noisy. We also provide quantitative results on FID in Appendix F.

Benchmarking `MonarchAttention`. Finally, we validate that the computational/IO complexity reduction achieved by `MonarchAttention` translates into actual speed-ups. We implement our kernels in Triton and compare with the fastest available implementation of `FlashAttention-2`. In the left panel of Figure 5, we sweep the sequence length N, showing that `MonarchAttention` consistently outperforms `FlashAttention-2`, notably achieving up to $8.2\times$ speed-up with $N = 16384$. For shorter sequences, we further fuse `MonarchAttention` to have a single thread block compute a single head of attention. In the right panel of Figure 5, we sweep the batch size at $N = 256$. With smaller batch sizes, we have low utilization of hardware. Yet `MonarchAttention` achieves up to $1.4\times$ speed-up over `FlashAttention-2` for larger batch sizes.

References

Mathieu Blondel, Andre Martins, and Vlad Niculae. Learning classifiers with fenchel-young losses: Generalized entropies, margins, and algorithms. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 606–615. PMLR, 2019.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. Scatterbrain: Unifying sparse and low-rank attention. *Advances in Neural Information Processing Systems*, 34:17413–17426, 2021.

Beidi Chen, Tri Dao, Kaizhao Liang, Jiaming Yang, Zhao Song, Atri Rudra, and Christopher Re. Pixelated butterfly: Simple and efficient sparse training for neural network models. International Conference on Learning Representations, 2022.

Theodore S Chihara. *An Introduction to Orthogonal Polynomials*. Courier Corporation, 2014.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. International Conference on Learning Representations, 2021.

Tri Dao, Albert Gu, Matthew Eichhorn, Atri Rudra, and Christopher Ré. Learning fast algorithms for linear transforms using butterfly factorizations. In *International conference on machine learning*, pages 1517–1527. PMLR, 2019.

Tri Dao, Beidi Chen, Nimit S Sohoni, Arjun Desai, Michael Poli, Jessica Grogan, Alexander Liu, Aniruddh Rao, Atri Rudra, and Christopher Ré. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, pages 4690–4721. PMLR, 2022a.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022b.

Giannis Daras, Nikita Kitaev, Augustus Odena, and Alexandros G Dimakis. Smyrf-efficient attention using asymmetric clustering. *Advances in Neural Information Processing Systems*, 33:6476–6489, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. International Conference on Learning Representations, 2021.

Dan Fu, Simran Arora, Jessica Grogan, Isys Johnson, Evan Sabri Eyuboglu, Armin Thomas, Benjamin Spector, Michael Poli, Atri Rudra, and Christopher Ré. Monarch mixer: A simple sub-quadratic gemm-based architecture. *Advances in Neural Information Processing Systems*, 36: 77546–77603, 2023.

Insu Han, R Jayaram, A Karbasi, V Mirrokno, D Woodruff, and A Zandieh. Hyperattention: Long-context attention in near-linear time. International Conference on Learning Representations, 2024.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Thomas Kailath, Sun-Yuan Kung, and Martin Morf. Displacement ranks of matrices and linear equations. *Journal of Mathematical Analysis and Applications*, 68(2):395–407, 1979.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. International Conference on Learning Representations, 2020.

Wojciech Kryściński, Nazneen Rajani, Divyansh Agarwal, Caiming Xiong, and Dragomir Radev. Booksum: A collection of datasets for long-form narrative summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 6536–6558, 2022.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, 2020.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, et al. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, 2021.

Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Victor Pan. *Structured matrices and polynomials: unified superfast algorithms*. Springer Science & Business Media, 2001.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4195–4205, 2023.

Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. cosformer: Rethinking softmax in attention. International Conference on Learning Representations, 2022.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International conference on machine learning*, pages 28492–28518. PMLR, 2023.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, 2016.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Ilya O Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Andreas Steiner, Daniel Keysers, Jakob Uszkoreit, et al. Mlp-mixer: An all-mlp architecture for vision. *Advances in neural information processing systems*, 34: 24261–24272, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

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

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Yunyang Xiong, Zhanpeng Zeng, Rudrasis Chakraborty, Mingxing Tan, Glenn Fung, Yin Li, and Vikas 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.

Michael Zhang, Kush Bhatia, Hermann Kumbong, and Christopher Re. The hedgehog & the porcupine: Expressive linear attentions with softmax mimicry. International Conference on Learning Representations, 2024.

Notation. We use $[N]$ to denote the index set $\{1, 2, \ldots, N\}$. We use Δ^N to denote the $(N-1)$ dimensional unit simplex, given by $\Delta^N = \{\boldsymbol{a} \in \mathbb{R}^N : \boldsymbol{a} \succeq 0, \langle \mathbf{1}_N, \boldsymbol{a} \rangle = 1\}$. We denote the $m \times m$ identity matrix by $\boldsymbol{I}_m$. We use the notation $\boldsymbol{A}_{ijk}$ to denote an element of a 3-way tensor, and $\boldsymbol{A}_{i,:,k}$ to denote a slice. We use δ_{kl} to denote the Kronecker delta that is 1 if $k = l$ and otherwise 0.

A. Related Work

Structured & Monarch Matrices. We use the phrase "structured matrices" to mean those that admit sub-quadratic storage and matrix-vector multiplication, such as low-rank or sparse matrices. There are many useful classes of structured matrices, such as those with *low displacement rank* (Kailath et al., 1979), which includes Toeplitz, Hankel, Vandermonde, Cauchy matrices (Pan, 2001); *orthogonal polynomial transforms* (Chihara, 2014), which includes discrete Fourier/cosine and Hadamard transforms; *butterfly factorizations* (Dao et al., 2019), which implement fast matrix-vector multiplication via a recursive divide-and-conquer algorithm similar to that of fast Fourier transforms (FFTs); and *Monarch matrices*, an expressive family of structured matrices (generalizing butterfly matrices and thereby many fast transforms) that overcome unfavorable memory access patterns typical to FFT-like algorithms by implementing matrix products via batched dense matrix multiplications (also called matmuls) on fast tensor cores found in modern GPUs.

Sub-Quadratic Attention.

- **Low-Rank.** Motivated by Johnson-Lindenstrauss embeddings, Wang et al. (2020) propose sketching the key and value matrices along the sequence dimension via learnable projections. Katharopoulos et al. (2020) introduce *linear attention*, where the exponential kernel is approximated via inner products of queries and keys lifted via some feature map. Several follow-up works proposed various feature maps, such as the exponential linear unit (ELU) (Katharopoulos et al., 2020), random positive features (Choromanski et al., 2021), rectified linear unit (ReLU) with cosine reweighting (Qin et al., 2022), and learnable single-layer multi-layer perceptrons (MLPs) (Zhang et al., 2024). Xiong et al. (2021) use the Nyström method for computing low-rank approximations by sampling rows and columns.
- **Sparse.** Child et al. (2019) introduce sparsity by applying fixed, structured sparse masks on the attention matrix. In particular, Chen et al. (2022) propose a particular *block* butterfly matrix for the sparse mask, which is more hardware-friendly at the cost of reduced expressiveness. Those that do not enforce a structure on the sparsity pattern include Kitaev et al. (2020); Daras et al. (2020) where they utilize locality-sensitive hashing (LSH) on shared query/key vectors to only compute attention within clusters of similar tokens.
- **Low-Rank + Sparse.** Inspired by robust PCA, Chen et al. (2021) decompose the attention matrix into a sum of two matrices: an unstructured sparse component using LSH and a low-rank component that is constructed via linear attention. Han et al. (2024) propose to subsample columns of the non-normalized attention matrix based on row norms of the value matrix, while estimating the softmax normalization factors from a few large elements via LSH.

Pure low-rank methods are often fast and hardware-friendly, but are not typically suitable as drop-in replacements for attention in pre-trained transformers due to the prevalence of "strongly diagonal", high-rank attention matrices where attention weights are concentrated locally in a sequence. Making up for this with a fixed sparsity pattern does not allow for data-dependent support of the attention matrix, necessary for zero-shot conversion. Finally, sparsity approaches that do not have a fixed sparsity pattern by relying on LSH improve on accuracy over low-rank approximations but suffer from significant overhead due to GPU incompatibility.

The next two works are closely related to our work.

- Dao et al. (2022b) propose `FlashAttention`, an IO-aware streaming algorithm for computing exact softmax attention. We show in Section 3 that each step of `MonarchAttention` can be written as a `FlashAttention`-like computation, allowing for similar IO savings to `FlashAttention` – in fact, we demonstrate that `MonarchAttention` achieves a strictly better worst-case IO complexity compared to `FlashAttention`. We also note that Dao et al. (2022b) propose to further accelerate `FlashAttention` using block butterfly attention masks, so `MonarchAttention` can be viewed as a generalization of block-sparse `FlashAttention` to more general Monarch matrices.
- `MonarchAttention` is closely related to Monarch Mixer (Fu et al., 2023), a mixer-type architecture (Tolstikhin et al., 2021) that utilizes Monarch instead of dense matrices for token and channel mixing. `MonarchAttention` also uses Monarch matrices for mixing tokens – however, it is based on the attention operation which is *data-dependent*, unlike Monarch Mixer.

B. Equivalence of Softmax Definitions

Consider the optimization problem in (2):

$$\max_{\boldsymbol{a}} \quad f(\boldsymbol{a}) := \sum_i \boldsymbol{a}_i \boldsymbol{z}_i - \sum_i \boldsymbol{a}_i \log \boldsymbol{a}_i \quad \text{s.t.} \quad \boldsymbol{a}_i \geq 0, \quad \sum_i \boldsymbol{a}_i = 1.$$

From the KKT stationarity condition, we have

$$\frac{\partial}{\partial \boldsymbol{a}_i}\left(f(\boldsymbol{a}) + \lambda\left(1 - \sum_i \boldsymbol{a}_i\right) + \sum_i \boldsymbol{\mu}_i \boldsymbol{a}_i\right) = 0 \implies \boldsymbol{z}_i - (1 + \log \boldsymbol{a}_i) - \lambda + \boldsymbol{\mu}_i = 0,$$

where $\lambda \in \mathbb{R}$, $\boldsymbol{\mu} \in \mathbb{R}^N$ are dual variables. From complementary slackness $\boldsymbol{a}_i \boldsymbol{\mu}_i = 0$ and the fact that $\log \boldsymbol{a}_i$ is not defined for $\boldsymbol{a}_i = 0$, we must have $\mu_i = 0$, which gives

$$\log \boldsymbol{a}_i = \boldsymbol{z}_i - \lambda - 1 \implies \boldsymbol{a}_i = \exp(\boldsymbol{z}_i)/\exp(\lambda + 1).$$

Finally, from the constraint $\sum_i \boldsymbol{a}_i = 1$, we must have $\exp(\lambda + 1) = \sum_j \exp(\boldsymbol{z}_j)$, which gives the form of softmax in (1).

C. Monarch Background

From (4), it follows that $\boldsymbol{B}$ has the following form:

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{L}_1^\top & & & \\ & \boldsymbol{L}_2^\top & & \\ & & \ddots & \\ & & & \boldsymbol{L}_b^\top \end{bmatrix} \boldsymbol{P} \begin{bmatrix} \boldsymbol{R}_1 & & & \\ & \boldsymbol{R}_2 & & \\ & & \ddots & \\ & & & \boldsymbol{R}_m \end{bmatrix},$$

where $\boldsymbol{L}_j \in \mathbb{R}^{m \times m}$ for $j \in [b]$ and $\boldsymbol{R}_k \in \mathbb{R}^{b \times b}$ for $k \in [m]$, and $\boldsymbol{P} \in \mathbb{R}^{N \times N}$ is a "transpose" permutation matrix whose $(i+1)$th row is given by $\boldsymbol{e}_{\sigma(i)+1}$ where

$$\sigma(i) = b \cdot (i \bmod m) + \left\lfloor \frac{i}{m} \right\rfloor, \quad i \in \{0, \ldots, N-1\}.$$

$\boldsymbol{P}\boldsymbol{x}$ corresponds to row-major reshaping $\boldsymbol{x} \in \mathbb{R}^N$ to $\mathbb{R}^{m \times b}$, transposing to $\mathbb{R}^{b \times m}$, then row-major flattening back to $\mathbb{R}^N$. As an illustrative example, let $N = 6$, $b = 3$, and $m = 2$. The action of $\boldsymbol{P}$ is given by the following steps:

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} \xrightarrow{\text{reshape } 2\times 3} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \xrightarrow{\text{transpose}} \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix} \xrightarrow{\text{flatten}} \begin{bmatrix} 1 \\ 4 \\ 2 \\ 5 \\ 3 \\ 6 \end{bmatrix}.$$

In matrix form, we have

$$\boldsymbol{P} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Given the above, matrix products are given by:

$$\boldsymbol{M}\boldsymbol{V} = \boldsymbol{O}, \quad \text{where} \quad \boldsymbol{O}_{b\cdot(l-1)+j,v} = \sum_k \boldsymbol{L}_{jkl} \boldsymbol{Y}_{jkv}, \quad \boldsymbol{Y}_{jkv} = \sum_i \boldsymbol{R}_{kji} \boldsymbol{V}_{b\cdot(k-1)+i,v}$$

for $i, j \in [b], k, l \in [m]$. A useful characterization of $\boldsymbol{M}$ is in block form:

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{M}_{11} & \dots & \boldsymbol{M}_{1m} \\ \vdots & \ddots & \vdots \\ \boldsymbol{M}_{m1} & \dots & \boldsymbol{M}_{mm} \end{bmatrix}, \tag{6}$$

where $\boldsymbol{M}_{lk} \in \mathbb{R}^{b \times b}$, $[\boldsymbol{M}_{lk}]_{ji} = \boldsymbol{L}_{jkl}\boldsymbol{R}_{kji}$, $\forall i, j \in [b], k, l \in [m]$.

D. Details for `MonarchAttention`

D.1. Updates

Derivatives. We evaluate f with $\boldsymbol{A} = \boldsymbol{M}$ as a Monarch matrix. Using (6), we have

$$\begin{aligned} f(\boldsymbol{M}; \boldsymbol{Q}, \boldsymbol{K}) &= \sum \boldsymbol{L}_{jkl}\boldsymbol{R}_{kji}\overline{\boldsymbol{Q}}_{jlv}\overline{\boldsymbol{K}}_{kiv} - \sum \boldsymbol{L}_{jkl}\boldsymbol{R}_{kji}\log(\boldsymbol{L}_{jkl}\boldsymbol{R}_{kji}) \\ &= \sum \boldsymbol{L}_{jkl}\boldsymbol{R}_{kji}\overline{\boldsymbol{Q}}_{jlv}\overline{\boldsymbol{K}}_{kiv} - \sum \boldsymbol{L}_{jkl}\boldsymbol{R}_{kji}\log \boldsymbol{R}_{kji} - \sum \boldsymbol{R}_{kji}\boldsymbol{L}_{jkl}\log \boldsymbol{L}_{jkl}. \end{aligned}$$

The derivatives of f w.r.t. each factor are given by

$$\frac{\partial f(\boldsymbol{M}; \boldsymbol{Q}, \boldsymbol{K})}{\partial \boldsymbol{L}_{jkl}} = \boldsymbol{\beta}_{L,jkl} - \boldsymbol{c}_{L,jk} - (1 + \log \boldsymbol{L}_{jkl})\boldsymbol{\gamma}_{L,jk}, \tag{7}$$

$$\frac{\partial f(\boldsymbol{M}; \boldsymbol{Q}, \boldsymbol{K})}{\partial \boldsymbol{R}_{kji}} = \boldsymbol{\beta}_{R,kji} - \boldsymbol{\gamma}_{R,kj} - (1 + \log \boldsymbol{R}_{kji})\boldsymbol{c}_{R,kj}, \tag{8}$$

where

$$\begin{aligned} \boldsymbol{\beta}_{L,jkl} &= \sum_v \overline{\boldsymbol{Q}}_{jlv} \sum_i (\boldsymbol{R}_{kji}\overline{\boldsymbol{K}}_{kiv}), \quad \boldsymbol{c}_{L,jk} = \sum_i \boldsymbol{R}_{kji}\log \boldsymbol{R}_{kji}, \quad \boldsymbol{\gamma}_{L,jk} = \sum_i \boldsymbol{R}_{kji}, \\ \boldsymbol{\beta}_{R,kji} &= \sum_v \overline{\boldsymbol{K}}_{kiv} \sum_l (\boldsymbol{L}_{jkl}\overline{\boldsymbol{Q}}_{jlv}), \quad \boldsymbol{\gamma}_{R,kj} = \sum_l \boldsymbol{L}_{jkl}\log \boldsymbol{L}_{jkl}, \quad \boldsymbol{c}_{R,kj} = \sum_l \boldsymbol{L}_{jkl}. \end{aligned}$$

We derive updates for each factor based on maximizing f with the other factor fixed.

$\boldsymbol{L}$ update. First, we fix $\boldsymbol{R} \in \Delta^{m \times b \times b}$ and consider

$$\max_{\boldsymbol{L}} \quad f(\boldsymbol{M}; \boldsymbol{Q}, \boldsymbol{K}) \quad \text{s.t.} \quad \boldsymbol{L}_{jkl} \geq 0, \quad \sum_k \boldsymbol{L}_{jkl} = 1.$$

From the KKT stationarity condition, we have

$$\frac{\partial}{\partial \boldsymbol{L}_{jkl}} \left(f(\boldsymbol{M}; \boldsymbol{Q}, \boldsymbol{K}) + \sum \boldsymbol{\lambda}_{L,jl} \left(1 - \sum_k \boldsymbol{L}_{jkl} \right) + \sum \boldsymbol{\mu}_{L,jkl}\boldsymbol{L}_{jkl} \right) = 0$$

where $\boldsymbol{\lambda}_L \in \mathbb{R}^{b \times m}$, $\boldsymbol{\mu}_L \in \mathbb{R}^{b \times m \times m}$ are dual variables. Along with (7), we have

$$\boldsymbol{\beta}_{L,jkl} - \boldsymbol{c}_{L,jk} - (1 + \log \boldsymbol{L}_{jkl})\boldsymbol{\gamma}_{L,jk} - \boldsymbol{\lambda}_{L,jl} + \boldsymbol{\mu}_{L,jkl} = 0.$$

Now, from complementary slackness $\boldsymbol{\mu}_{L,jkl}\boldsymbol{L}_{jkl} = 0$ and the fact that $\log \boldsymbol{L}_{jkl}$ is not defined for $\boldsymbol{L}_{jkl} = 0$, we must have $\boldsymbol{\mu}_{L,jkl} = 0$. Moreover, since $\boldsymbol{R} \in \Delta^{m \times b \times b}$, we have $\boldsymbol{\gamma}_{L,jk} = 1$. Altogether, we have

$$\log \boldsymbol{L}_{jkl} = \boldsymbol{\beta}_{L,jkl} - \boldsymbol{c}_{L,jk} - \boldsymbol{\lambda}_{L,jl} - 1 \implies \boldsymbol{L}_{jkl} = \frac{\exp(\boldsymbol{\beta}_{L,jkl} - \boldsymbol{c}_{L,jk})}{\exp(\boldsymbol{\lambda}_{L,jl} + 1)}.$$

Finally, from the constraint $\sum_k \boldsymbol{L}_{jkl} = 1$, we must have $\exp(\boldsymbol{\lambda}_{L,jl} + 1) = \sum_k \exp(\boldsymbol{\beta}_{L,jkl} - \boldsymbol{c}_{L,jk})$, which gives the final closed form update:

$$\boldsymbol{L} = \mathrm{softmax}_k(\boldsymbol{Z}_L), \quad \boldsymbol{Z}_{L,jkl} = \boldsymbol{\beta}_{L,jkl} - \boldsymbol{c}_{L,jk},$$

where $\mathrm{softmax}_k$ is applied along the k index dimension.

$\boldsymbol{R}$ update. Similarly, we fix $\boldsymbol{L} \in \Delta^{b\times m\times m}$ and consider

$$\max_{\boldsymbol{R}} \quad f(\boldsymbol{M};\boldsymbol{Q},\boldsymbol{K}) \quad \text{s.t.} \quad \boldsymbol{R}_{kji} \geq 0, \quad \sum_i \boldsymbol{R}_{kji} = 1.$$

From the KKT stationarity condition, we have

$$\frac{\partial}{\partial \boldsymbol{R}_{kji}} \left(f(\boldsymbol{M};\boldsymbol{Q},\boldsymbol{K}) + \sum \boldsymbol{\lambda}_{R,kj} \left(1 - \sum_i \boldsymbol{R}_{kji}\right) + \sum \boldsymbol{\mu}_{R,kji}\boldsymbol{R}_{kji} \right) = 0$$

where $\boldsymbol{\lambda}_R \in \mathbb{R}^{m\times b}$, $\boldsymbol{\mu}_R \in \mathbb{R}^{m\times b\times b}$ are dual variables. Along with (8), we have

$$\boldsymbol{\beta}_{R,kji} - \boldsymbol{\gamma}_{R,kj} - (1+\log \boldsymbol{R}_{kji})\boldsymbol{c}_{R,kj} - \boldsymbol{\lambda}_{R,kj} + \boldsymbol{\mu}_{R,kji} = 0.$$

As before, from complementary slackness $\boldsymbol{\mu}_{R,kji}\boldsymbol{R}_{kji} = 0$ and the fact that $\log \boldsymbol{R}_{kji}$ is not defined for $\boldsymbol{R}_{kji} = 0$, we must have $\boldsymbol{\mu}_{R,kji} = 0$. Altogether, we have

$$\log \boldsymbol{R}_{kji} = \frac{\boldsymbol{\beta}_{R,kji} - \boldsymbol{\gamma}_{R,kj} - \boldsymbol{\lambda}_{R,kj}}{\boldsymbol{c}_{R,kj}} - 1 \implies \boldsymbol{R}_{kji} = \frac{\exp(\boldsymbol{\beta}_{R,kji}/\boldsymbol{c}_{R,kj})}{\exp(\boldsymbol{\gamma}_{R,kj}/\boldsymbol{c}_{R,kj} + \boldsymbol{\lambda}_{R,kj}/\boldsymbol{c}_{R,kj} + 1)}.$$

Finally, from the constraint $\sum_i \boldsymbol{R}_{kji} = 1$, we must have $\exp(\boldsymbol{\gamma}_{R,kj}/\boldsymbol{c}_{R,kj} + \boldsymbol{\lambda}_{R,kj}/\boldsymbol{c}_{R,kj} + 1) = \sum_i \exp(\boldsymbol{\beta}_{R,kji}/\boldsymbol{c}_{R,kj})$, which gives the final closed form update:

$$\boldsymbol{R} = \mathrm{softmax}_i(\boldsymbol{Z}_R), \quad \boldsymbol{Z}_{R,kji} = \boldsymbol{\beta}_{R,kji}/\boldsymbol{c}_{R,kj},$$

where $\mathrm{softmax}_i$ is applied along the i index dimension.

D.2. Naïve Algorithm

We provide pseudo-code for the naive version of `MonarchAttention` in Figure 6. This is a direct implementation of alternating maximization for finding the Monarch factors, ignoring concerns about memory and IO. We alternate between updating $\boldsymbol{L}$ and $\boldsymbol{R}$ as described above for T iterations. Specifically, we highlight the choices to (1) initialize $\boldsymbol{L}$ to be block identity, and (2) update $\boldsymbol{R}$ before $\boldsymbol{L}$ in each iteration.

D.3. Padding and Masking

In practice, the sequence length N may not be divisible by the desired block size b. In such cases, we round the number of blocks m to $m' = \lceil N/b \rceil$, and set the new sequence length $N' = m'b$, post-padding $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ to have N' rows. However, we need to take special care that the final $N' - N$ columns of the padded Monarch attention matrix $\boldsymbol{M} \in \Delta^{N'\times N'}$ are zero, since these correspond to padded rows of $\boldsymbol{V}$. This is also an issue when batched sequences of different lengths are padded to a maximum length to avoid dynamic resizing.

From (6), it is clear that to set all columns of $\boldsymbol{M}$ beyond the Nth column to zero, it is sufficient to set $\boldsymbol{R}_{kji}$ to zero whenever $b(k-1)+i > N$. Thus, we simply form the mask $\boldsymbol{\omega} \in \mathbb{R}^{m'\times b\times b}$ given by

$$\boldsymbol{\omega}_{kji} = \begin{cases} 0 & b(k-1)+i \leq N \\ -\infty & \text{otherwise,} \end{cases}$$

which we then add to $\boldsymbol{Z}_R$ before softmax in (**??**). We can also pre-pad the sequence, which would be change the above condition to $b(k-1)+i \geq N' - N$.

D.4. Implementation

To minimize data movement and memory usage on GPU, we do not materialize $\boldsymbol{L}$ or $\boldsymbol{R}$ in high-bandwidth memory (HBM). In addition to $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}, \boldsymbol{O}$, we only need to maintain states[2] $\boldsymbol{\alpha}_R^{(t)}, \boldsymbol{\alpha}_L^{(t)}, \boldsymbol{c}_R^{(t)}, \boldsymbol{c}_L^{(t)}$ from (**??**) and (**??**), resulting in $\Theta(Nd)$

[2]The $\boldsymbol{\alpha}$ and $\boldsymbol{c}$ variables can share the same memory location as those corresponding to $\boldsymbol{R}$ can be derived from $\boldsymbol{L}$ (and vice-versa).

```
# Q: array of size (N, d)
# K: array of size (N, d)
# V: array of size (N, d)
# T: number of steps

def monarch_attention(Q, K, V, T):
    L = stack(b * [eye(m)])
    Qb = einshape("(lj)v->jlv", Q, j=b)
    Kb = einshape("(ki)v->kiv", K, i=b)

    # Alternating maximization for L, R
    for t in range(T):
        # R update
        aR = einsum("jkl,jlv->kjv", L, Qb)
        bR = einsum("kjv,kiv->kji", aR, Kb)
        cR = einsum("jkl->kj", L)
        R = softmax(bR / cR[:, :, None], axis=2)

        # L update
        aL = einsum("kji,kiv->jkv, R, Kb)
        bL = einsum("jkv,jlv->jkl", aL, Qb)
        cL = einsum("kji->jk", R * log(R))
        L = softmax(bL - cL[:, :, None], axis=1)

    # Monarch multiply
    Vb = einshape("(ki)v->kiv", V, i=b)
    Y = einsum("kji,kiv->jkv", R, Vb)
    Z = einsum("jkl,jkv->ljv, L, Y)
    O = einshape("ljv->(lj)v", Z)

    return O
```

Figure 6: Naïve implemention of `MonarchAttention`

additional memory. All other intermediate values are only materialized in on-chip SRAM, fusing all operations between the $\boldsymbol{\alpha}$ (and similarly $\boldsymbol{c}$) variables. For instance, from the above update equations, the computation of $\boldsymbol{\alpha}_L^{(t)}$ from $\boldsymbol{\alpha}_R^{(t)}$ is given by

$$\boldsymbol{\alpha}_{L,jkv}^{(t)} = \mathrm{softmax}_i\left(\frac{\sum_v \boldsymbol{\alpha}_{R,kjv}^{(t)} \overline{\boldsymbol{K}}_{kiv}}{\boldsymbol{c}_{R,kj}^{(t)}}\right) \overline{\boldsymbol{K}}_{kiv},$$

which can be seen as a batched attention computation, meaning we can implement a `FlashAttention`-like kernel to reduce IO between HBM and on-chip SRAM. However, several aspects of this computation make it particularly IO-efficient. Besides the fact that $\overline{\boldsymbol{K}}$ acts as both the $\boldsymbol{K}$ and $\boldsymbol{V}$ matrices in (3), the effective sequence length is $\sqrt{N}$. This eliminates the need for tiling along the sequence length, except for very long sequences having $\Theta(\sqrt{N}d) > S$, where S is the size of on-chip SRAM. This means that we have an optimal IO complexity of $\Theta(Nd)$ for a single call, as opposed to the worst-case $O(N^2d^2/S)$ complexity of `FlashAttention`. The computation of $\boldsymbol{\alpha}_R^{(t)}$ from $\boldsymbol{\alpha}_L^{(t)}$, as well as the Monarch matmul, can be written in a similar fashion. Python-like code for `MonarchAttention` is given in Figure 7.

E. Experimental Details

E.1. Baselines

We describe the baselines used in Section 4. We specifically exclude low-rank methods with learnable components (Wang et al., 2020; Zhang et al., 2024), since we are focused on the zero-shot setting, as well as sparsity/LSH-based approaches (Kitaev et al., 2020; Daras et al., 2020; Chen et al., 2021; Han et al., 2024), since these do not admit efficient implementations on current GPUs.

- `linear-attention` (Katharopoulos et al., 2020) approximates $\exp(\boldsymbol{q}^\top \boldsymbol{k}) \approx \phi(\boldsymbol{q})^\top \phi(\boldsymbol{k})$ where $\phi : \mathbb{R}^d \to \mathbb{R}^r$ is a

```python
def al_cl_kernel(aR, cR, Kb): # Computes aL, cL from aR, cR
    R = softmax(bmm(aR, Kb.transpose(1, 2)) / cR[:, :, None], dim=2)
    cL = sum(R * log(R), dim=2).transpose(0, 1)
    aL = bmm(R, Kb).transpose(0, 1)
    return aL, cL

def ar_cr_kernel(aL, cL, Qb): # Computes aR, cR from aL, cL
    L = softmax(bmm(aL, Qb.transpose(1, 2)) - cL[:, :, None], dim=1)
    cR = sum(L, dim=2).transpose(0, 1)
    aR = bmm(L, Qb).transpose(0, 1)
    return aR, cR

def al_y_cl_kernel(aR, cR, Kb, Vb): # Fuse al_cl_kernel + Monarch matmul 1st step
    R = softmax(bmm(aR, Kb.transpose(1, 2)) / cR[:, :, None], dim=2)
    cL = sum(R * log(R), dim=2).transpose(0, 1)
    aL = bmm(R, Kb).transpose(0, 1)
    y = bmm(R, Vb).transpose(0, 1)
    return aL, y, cL

def z_kernel(aL, y, cL, Qb): # Monarch matmul 2nd step
    L = softmax(bmm(Qb, aL.transpose(1, 2)) - cL[:, None, :], dim=2)
    z = bmm(L, y).transpose(0, 1)
    return z

def monarch_attention(Q, K, V, T): # Q, K, V: (N, d), T: number of steps
    Qb = Q.reshape(m, b, d).transpose(0, 1)
    Kb = K.reshape(m, b, d)
    Vb = V.reshape(m, b, d)
    aR = Q.reshape(m, b, d)
    cR = ones(m, b)

    for t in range(T-1):
        aL, cL = al_cl_kernel(aR, cR, Kb)
        aR, cR = ar_cr_kernel(aL, cL, Qb)

    aL, y, cL = al_y_cl_kernel(aR, cR, Kb, Vb)
    z = z_kernel(aL, y, cL, Qb)
    o = z.reshape(N, d)
    return o
```

Figure 7: **Python-like code for `MonarchAttention`.** Each kernel materializes all intermediate arrays in SRAM to reduce data movement.

kernel feature map, resulting in a rank r approximation to softmax attention:

$$\mathrm{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top) \approx \frac{\phi(\boldsymbol{Q})\phi(\boldsymbol{K})^\top}{\phi(\boldsymbol{Q})\phi(\boldsymbol{K})^\top \mathbf{1}_N \mathbf{1}_N^\top}.$$

Katharopoulos et al. (2020) propose the map $\phi(\boldsymbol{x}) = 1 + \mathrm{elu}(\boldsymbol{x})$ with $r = d$ where elu is the exponential linear unit applied element-wise.

- `performer` (Choromanski et al., 2021) is a linear attention method using the fact that

$$\exp(\boldsymbol{q}^\top \boldsymbol{k}) = \mathbb{E}_{\boldsymbol{\omega}\sim\mathcal{N}(\mathbf{0},\boldsymbol{I}_d)}\left[\exp\left(\boldsymbol{\omega}^\top \boldsymbol{q} - \frac{\|\boldsymbol{q}\|^2}{2}\right)\exp\left(\boldsymbol{\omega}^\top \boldsymbol{k} - \frac{\|\boldsymbol{k}\|^2}{2}\right)\right]$$

to construct a random kernel feature map

$$\phi(\boldsymbol{x}) = \frac{1}{\sqrt{r}}\exp\left(-\frac{\|\boldsymbol{x}\|^2}{2}\right)\begin{bmatrix}\exp(\boldsymbol{\omega}_1^\top \boldsymbol{x}) & \dots & \exp(\boldsymbol{\omega}_r^\top \boldsymbol{x})\end{bmatrix}^\top,$$

where $\boldsymbol{\omega}_1, \dots, \boldsymbol{\omega}_r \overset{iid}{\sim} \mathcal{N}(\mathbf{0}, \boldsymbol{I}_d)$.

- `cosformer` (Qin et al., 2022) is a linear attention method utilizing position-dependent kernel feature maps of the form

$$\phi_i(\boldsymbol{x}) = \left[\sin\left(\tfrac{\pi i}{2N}\right)\text{relu}(\boldsymbol{x}_i) \quad \cos\left(\tfrac{\pi i}{2N}\right)\text{relu}(\boldsymbol{x}_i)\right], \ \forall i \in [N],$$

  which produces a rank $r = 2d$ approximation.

- `nystromformer` (Xiong et al., 2021) computes landmark $\tilde{\boldsymbol{Q}}, \tilde{\boldsymbol{K}} \in \mathbb{R}^{r\times d}$ from $\boldsymbol{Q}, \boldsymbol{K}$ by averaging N/r consecutive spans of rows, which are used to approximate softmax attention via the quadrature method:

$$\tilde{\boldsymbol{F}} = \text{softmax}(\boldsymbol{Q}\tilde{\boldsymbol{K}}^\top), \ \tilde{\boldsymbol{B}} = \text{softmax}(\tilde{\boldsymbol{Q}}\boldsymbol{K}^\top), \ \tilde{\boldsymbol{A}} = \text{softmax}(\tilde{\boldsymbol{Q}}\tilde{\boldsymbol{K}}^\top)$$
$$\text{softmax}(\boldsymbol{Q}\boldsymbol{K}^\top) \approx \tilde{\boldsymbol{F}}\tilde{\boldsymbol{A}}^+\tilde{\boldsymbol{B}},$$

  where $\tilde{\boldsymbol{A}}^+$ denotes the pseudoinverse of $\tilde{\boldsymbol{A}}$, producing a rank r approximation.

E.2. Image Classification with ViT

We convert all 12 attention layers, each having 12 heads with sequence length $N = 197$ and head dimension $d = 64$, of the 87M parameter ViT-B (Dosovitskiy et al., 2021) that has been pre-trained on ImageNet-21K (Deng et al., 2009) and fine-tuned on ImageNet-1K (Russakovsky et al., 2015) for image classification. To evaluate the performance at different FLOP counts, we vary the number of steps T for `MonarchAttention`, and vary the rank for Performer and Nyströmformer as follows:

- `monarch-attention`: $b = 14$ and $T \in \{1, 2, 3\}$
- `performer`: $r \in \{16, 32, 48, 64, 80, 96\}$
- `nystromformer`: $r \in \{16, 24, 32, 40\}$

The ViT-B model fine-tuned on ImageNet-21K is retrieved from the Hugging Face `transformers` library (Wolf et al., 2019) as `google/vit-base-patch16-224`. The ImageNet-1K evaluation dataset is retrieved from the Hugging Face `datasets` library (Lhoest et al., 2021) as `imagenet-1k` using the `validation` split.

E.3. Question Answering with RoBERTa

We convert the initial 4 and final 4 layers of the 12 attention layers, each having 12 heads with sequence length $N = 384$ and head dimension $d = 64$, of the 125M parameter RoBERTa-B (Liu et al., 2019) that has been pre-trained on a large English corpus and fine-tuned on SQuAD1.1 (Rajpurkar et al., 2016) for question answering. As before, to evaluate the performance at different FLOP counts, we vary the block size b for `MonarchAttention`, and vary the rank for Performer and Nyströmformer as follows:

- `monarch-attention`: $T = 1$ and $b \in \{24, 48, 96, 128\}$
- `performer`: $r \in \{32, 64, 96, 128, 160, 192\}$
- `nystromformer`: $r \in \{16, 32, 48, 64\}$

The RoBERTa-B model fine-tuned on SQuAD1.1 is retrieved from the Hugging Face `transformers` library as `csarron/roberta-base-squad-v1`. The SQuAD1.1 evaluation dataset is retrieved from the Hugging Face `datasets` library as `squad` using the `validation` split. For evaluation, we truncate and pad to sequence length of 384.

E.4. Summarization with BART

We convert all 6 attention layers, each having 12 heads with head dimension $d = 64$, in the encoder of the 139M parameter BART-B (Lewis et al., 2020) that has been pre-trained on a large English corpus and fine-tuned on BookSum-chapters (Kryściński et al., 2022) for summarization. We only convert the encoder model and leave the decoder intact. To evaluate

Table 1: Hyperparameters used for BART summarization.

Sequence Length	Method	(b, T)	r	Total Attention FLOPs (10^9)
1024	Softmax	–	–	9.66
	Nyströmformer	–	64	1.93
	`MonarchAttention`	(32, 3)	–	1.96
2048	Softmax	–	–	38.7
	Nyströmformer	–	80	4.41
	`MonarchAttention`	(32, 2)	–	3.93
4096	Softmax	–	–	155.
	Nyströmformer	–	112	10.6
	`MonarchAttention`	(64, 2)	–	10.9
8192	Softmax	–	–	619.
	Nyströmformer	–	160	35.0
	`MonarchAttention`	(64, 2)	–	31.4

the benefits of sub-quadratic attention for processing longer sequences, we truncate the text to be summarized to various sequence lengths N for each method. The hyperparameters for each method across sequence lengths are shown in Table 1.

The pre-trained BART-B model is retrieved from the Hugging Face `transformers` library as `facebook/bart-base`. The BookSum-chapters training/evaluation dataset is retrieved from the Hugging Face `datasets` library as `kmfoda/booksum` using the `train` and `validation` splits respectively. BART employs learned positional embeddings up to 1024 sequence length, and since we are interested in long-sequence summarization up to 8192 tokens, we linearly interpolate the encoder positional embeddings up to 8192 tokens, before fine-tuning on BookSum-chapters – we leave the decoder positional embeddings intact. We fine-tune for 5 epochs with batch size of 32 and learning rate of 10^{-4} using the Adam optimizer (Kingma and Ba, 2014) without weight decay, with the input and summary sequences truncated and padded to 8192 and 512 tokens respectively. For evaluation, we truncate the input sequence to the corresponding sequence length in Figure 2.

E.5. Image Generation with DiT

We convert a subset of the 28 attention layers, each having 16 heads with sequence length $N = 256$ and head dimension $d = 72$, of the 675M parameter DiT-XL (Peebles and Xie, 2023) that has been trained on ImageNet (Deng et al., 2009). We consider replacing either all layers, the first 14 layers, or the last 14 layers; see Appendix E.5 for more details on the set-up. Examples of generated images with each method for replacing the first 14 layers are shown in Figure 4, where `MonarchAttention` produces clear images resembling those of softmax attention, while the Nyströmformer ones are extremely noisy. We also provide quantitative results on FID in Appendix F.

The pre-trained model DiT-XL is retrieved from the Hugging Face `transformers` library as `facebook/DiT-XL-2-256`. Following Peebles and Xie (2023), we generate images using 32 sampling steps, a 2×2 patch size, and a classifier-free guidance scale of 1.5. We use the following hyperparameters:

- `monarch-attention`: $b = 16$ and $T = 3$
- `nystromformer`: $r = 32$

F. Additional DiT Results

Here we quantitatively evaluate the (s)FID scores (Heusel et al., 2017) of `MonarchAttention` compared with Nyströmformer, using images generated with the original softmax attention model as reference – the results are reported in Table 2. `MonarchAttention` noticeably outperforms Nyströmformer with similar FLOP count. In particular, using `MonarchAttention` in the first half of the DiT layers results in extremely small FID and sFID from the softmax attention model's images, while reducing FLOPs by nearly 30%.

Table 2: **Quantitative results for zero-shot conversion of attention layers for image generation.** We report FID and sFID (using the original softmax attention model as reference) of DiT when replacing all or half of the attention layers.

Layers Replaced	**Method**	**Total Attention FLOPs** (10^9)	**FID** (↓)	**sFID** (↓)
–	Softmax	8.46	–	–
All	Nyströmformer	3.30	5.97	13.47
	`MonarchAttention`	3.44	**2.82**	**5.09**
First Half	Nyströmformer	5.88	8.17	19.01
	`MonarchAttention`	5.95	**0.39**	**0.66**
Second Half	Nyströmformer	5.88	6.76	13.58
	`MonarchAttention`	5.95	**1.98**	**3.36**