SIMPLE HARDWARE-EFFICIENT LONG CONVOLUTIONS FOR SEQUENCE MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

State space models (SSMs) have high performance on long sequence modeling but require sophisticated initialization techniques and specialized implementations for high quality and runtime performance. We study whether a simple alternative can match SSMs in performance and efficiency: directly learning long convolutions over the sequence. We find that simply squashing the long convolutional kernel weights is enough to match SSMs in performance on a range of tasks including the long range arena (LRA) and language modeling. To also improve runtime performance, we next develop FLASHBUTTERFLY, an IO-aware algorithm to compute long convolutions efficiently. FLASHBUTTERFLY appeals to classic Butterfly decompositions of the convolution to reduce GPU memory IO and increase FLOP utilization. FLASHBUTTERFLY speeds up the LRA benchmark by 7.0× over Transformers, and allows us to train on Path256, a challenging task with sequence length 64K, where we set state-of-the-art by 29.1 points while training 7.2× faster than prior work.

1 INTRODUCTION

A fundamental question in understanding foundation models is whether their success depends on specific architectures like attention, or whether simpler alternatives can also suffice. Recently, a new class of sequence models based on state space models (SSMs) (Gu et al., 2022a; Li et al., 2022; Hasani et al., 2022; Gupta et al., 2022) has emerged as a powerful general-purpose sequence modeling framework. SSMs scale nearly linearly in sequence length and have shown state-of-the-art performance on a range of sequence modeling tasks, from long range sequence modeling (Smith et al., 2022) to language modeling (Dao et al., 2022c; Ma et al., 2022).

However, SSMs rely on sophisticated mathematical structures to train effectively in deep networks (Gu et al., 2022a). These structures generate a convolution kernel as long as the input sequence by repeatedly multiplying a hidden *state* matrix. This process may be unstable (Goel et al., 2022) and requires careful hand-crafted initializations (Gu et al., 2022b), leaving practitioners with a dizzying array of choices and hyperparameters. In this paper, we study whether we can replace the SSMs with an even simpler approach – parameterizing the long convolution kernel directly.

There are two challenges that long convolutions face for sequence modeling. The first is quality: previous attempts at directly parameterizing the convolution kernel have underperformed SSMs (Romero et al., 2021; Li et al., 2022). The second is runtime efficiency: long convolutions can be computed in $O(N \log N)$ FLOPS in sequence length N using the Fast Fourier transform (FFT), but systems constraints often make them slower than quadratic algorithms, such as attention. In this paper, we show that a simple regularization technique and an IO-aware convolution algorithm can address these challenges.

Closing the Quality Gap. First, to understand the quality gap, we study the performance of long convolutions compared to SSMs on Long Range Arena (LRA) (Tay et al., 2020), a key benchmark designed to test long sequence models. Long convolutions underperform SSMs by up to 16.6 points on average (Table 1). We find a simple regularization technique using a SQUASH operator to reduce the magnitude of the kernel weights. Using this regularization, long convolutions also appear more robust to initialization than SSMs, matching S4 on LRA even with *completely random* initialization.

We further evaluate the performance of long convolutions on text modeling, where they are competitive with the recent H3 model (Dao et al., 2022c)—coming within 0.3 PPL on OpenWebText—and outperform Transformers by 0.7 PPL on OpenWebText.

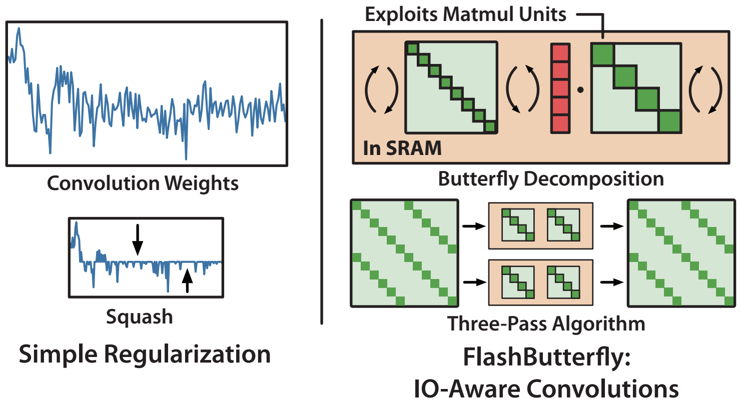

Figure 1: Left: A Simple regularization technique allow long convolutions to match state space models in sequence modeling. Right: We develop FLASHBUTTERFLY, an IO-aware algorithm for long convolutions.

Closing the Runtime Performance Gap. However, long convolutions are inefficient on modern hardware, since the FFT convolution incurs expensive GPU memory IO and cannot utilize matrix multiply units—even using optimized implementations like cuFFT (NVIDIA, 2022). SSM convolution formulations rely on specialized GPU Cauchy kernels and log Vandermonde, as well as special recurrent message passing structure, to overcome these challenges.

In response, we develop FLASHBUTTERFLY, a simple IO-aware algorithm for long convolutions, which does not require ad hoc hand engineering. FLASHBUTTERFLY appeals to classic Butterfly decompositions of the FFT to rewrite the FFT convolution as a series of block-sparse Butterfly matrices. This decomposition reduces the number of passes over the input sequence—reducing the GPU memory requirements—and utilizes matrix multiply units on the GPU, which increases FLOP utilization.

To demonstrate FLASHBUTTERFLY's scaling ability, we train a long convolution model on Path256, a task with sequence length 64K. We set state-of-the-art by 29.1 points and train 7.2× faster than the previous best model.

Summary. In summary, we show that long convolutions are an effective model for long sequence modeling. They match or exceed SSMs across an array of diverse sequence domains while requiring less hand-crafted initializations and showing improved stability. Additionally, by leveraging connections to Butterfly matrices, long convolutions can be trained up to 2.2× faster than SSMs.

2 BACKGROUND

Deep State Space Models A discrete-time state space model (SSM) linearly maps an input $u \in \mathbb{R}^N$, over time $t \in \{1,...,N\}$, to an output signal $y \in \mathbb{R}^N$ as $x_t = \mathbf{A}x_{t-1} + \mathbf{B}u_t$, $y_t = \mathbf{C}x_t + \mathbf{D}u_t$, by the use of hidden state $x_t \in \mathbb{R}^d$ and some set of matrices $\mathbf{A} \in \mathbb{R}^{d\times d}$, $\mathbf{D} \in \mathbb{R}^{1\times 1}$, $\mathbf{B} \in \mathbb{R}^{d\times 1}$, $\mathbf{C} \in \mathbb{R}^{1\times d}$. By unrolling the recursion, y can be written as a convolution between u and a kernel $\mathbf{K}$ that depends on $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$: $y = \mathbf{K} * u + \mathbf{D}u$. Deep SSM models often contain several stacked SSM blocks, each of which is comprised of H heads of parallel SSMs with independent learnable parameters.

Long Convolutions as Sequence Models Rather than parameterizing $\mathbf{K}$ with carefully initialized SSM matrices, we seek to directly parameterize the convolution kernel $\mathbf{K}$. Our goal is to replace the SSM layer with a learned convolution kernel as a drop-in replacement, while keeping the stacking and multi-head structure of SSM models (which can be thought of as multiple convolutional filters).

FFT Convolution A standard approach to compute convolutions in $O(N \log N)$ in sequence length N is to use the FFT convolution theorem. Let $\mathbf{F}_N$ denote the DFT matrix of size N. Then, the convolution can be computed as: $y = u * \mathbf{K} = \mathbf{F}_N^{-1}\mathbf{D}_{\mathbf{K}}\mathbf{F}_N u$, where $\mathbf{D}_{\mathbf{K}} = \text{diag}(\mathbf{F}_N \mathbf{K})$.

3 METHOD

In Section 3.1, we conduct an initial investigation into long convolutions for sequence modeling, and develop a simple regularization strategy based on our findings. Then, in Section 3.2, we present FLASHBUTTERFLY, an IO-aware algorithm for speeding up convolutions using a connection to block-sparse matrix multiplication.

3.1 LONG CONVOLUTIONS FOR SEQUENCE MODELING

First, we conduct a brief investigation into the performance of long convolutions on sequence modeling, and we find a gap in quality. We then propose a simple regularization technique for closing this gap.

Regularizing the Kernel. We begin by directly replacing the SSM layers in an S4 model with long convolutions, with random initialization. Table 1 shows that long convolutions underperform SSMs by 16.6 points on average across LRA. We propose a simple technique for regularizing the convolution kernel using the SQUASH operator. The SQUASH operator is applied element-wise to the convolution kernel, and reduces the magnitude of all weights: $\overline{\mathbf{K}} = \text{sign}(\mathbf{K}) \odot \max(|\mathbf{K}| - \lambda, 0)$.

Initialization. To understand the impact of initialization on long convs, we evaluate long convs on two simple intialization techniques: random initialization, and a geometric decay. The random initialization initializes the weights to be randomly distributed from a Normal distribution: $\mathbf{K}_i \sim \mathcal{N}$. The geometric decay initialization additionally scales kernel weights to decay across the sequence, as well as across the heads. For the kernel $\mathbf{K}^{(h)}$, $1 \leq h \leq H$, we initialize the weights as: $\mathbf{K}_k^{(h)} = x\exp(-kN^{-1}(H/2)^{hH^{-1}})$, for $1 \leq k \leq N$, where $x \sim \mathcal{N}$ is drawn from a Normal distribution.

3.2 FLASHBUTTERFLY

We present FLASHBUTTERFLY, an IO-aware algorithm for speeding up general convolutions on modern hardware. Following H3 (Dao et al., 2022c), we use kernel fusion to reduce GPU memory IO requirements, and use a Butterfly decomposition to rewrite the FFT as a series of block-sparse matrix multiplications, allowing better utilization of modern matrix multiply units. The details are shown in Appendix C. To scale to sequences that does not fit into SRAM (length 8K or longer on A100), the method presented in H3 (Dao et al., 2022c) does not work anymore, as it depended in a critial way on the recurrent nature of convolutions induced by SSMs. Instead, we use an alternate Butterfly decomposition to construct a three-pass FFT convolution algorithm to further reduce IO requirements.

Three-Pass Algorithm. We exploit two alternative formulations of the Butterfly decomposition of the FFT. A DFT matrix $\mathbf{F}_N$ of size N can be written as $N\mathbf{P}^{-1}(\mathbf{I}_m \otimes (l\mathbf{F}_l))\overline{\mathbf{B}}^{-1}$, and its inverse matrix $\mathbf{F}_N^{-1}$ can be written as $N^{-1}\overline{\mathbf{B}}(\mathbf{I}_m \otimes \overline{\mathbf{F}}_l)\mathbf{P}$, where $\mathbf{B}$ is an $N \times N$ block matrix with m^2 blocks of size $l \times l$, each of which is diagonal (see Appendix C for the exact derivation). Critically, matrix-vector multiply $\mathbf{B}u$ can be computed in a single pass over the input vector u. Substituting these into the FFT convolution decomposition and simplifying yields the following: $y = u * \mathbf{K} = \overline{\mathbf{B}}(\mathbf{I}_m \otimes \overline{\mathbf{F}}_l)\mathbf{D}'_{\mathbf{K}}(\mathbf{I}_m \otimes \mathbf{F}_l)\overline{\mathbf{B}}^{-1}$, where $\mathbf{D}'_{\mathbf{K}} = l\mathbf{P}\mathbf{D}_{\mathbf{K}}\mathbf{P}^{-1}$ is another diagonal matrix. The middle terms can now be computed as m independent FFT convolutions of size l, with a different convolution kernel. These parallel convolutions collectively require one pass over N input elements, so the entire convolution can be computed with three passes over the input.

4 EVALUATION

We evaluate how well long convolutions perform in the challenging LRA benchmark as well as on the OpenWebText language task. Next, we evaluate the runtime efficiency of long convolutions under FLASHBUTTERFLY and evaluate how well it scales to very long sequences (Section 4.2).

4.1 QUALITY ON SEQUENCE MODELING

We begin by evaluating various regularization and initialization techniques on the long range arena benchmark, a suite of six general-purpose sequence modeling tasks with sequence length between 1K and 16K tokens, covering modalities including text, natural and synthetic images, and mathematical expressions (Tay et al., 2020). We then evaluate long convolutions on language modeling. Experimental details for the tasks are given in Appendix E, and additional experiments are provided in Appendix B.

Long Sequence Modeling: Long Range Arena. Table 1 shows the results for long convolutions on the LRA benchmark. An ✗ in the Path-X column indicates that the model never achieved better

Table 1: Validation accuracy of different models on the LRA benchmark. Best in bold, second best underlined.

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Transformer	36.4	64.3	57.5	42.4	71.4	✗	53.7
S4-LegS	59.6	86.8	<u>90.9</u>	<u>88.7</u>	94.2	<u>96.4</u>	86.1
S4-FouT	57.9	86.2	89.7	**89.1**	<u>94.5</u>	✗	77.9
S4 (Original)	58.4	76.0	87.1	87.3	86.1	88.1	80.5
Long Convs	53.4	64.4	83.0	81.4	85.0	✗	69.5
Long Convs, Random Init + SQUASH	<u>61.4</u>	<u>88.0</u>	90.2	<u>88.7</u>	**94.6**	**97.1**	**86.7**
Long Convs, Exp Init + SQUASH	**62.2**	**89.6**	**91.3**	87.0	93.2	96.0	<u>86.6</u>

Table 2: Test PPL of models trained on OpenWebText.

Model	Test PPL
Transformer	20.6
GSS	24.0
H3	**19.6**
H3 + Long-Conv, Rand Init	20.1
H3 + Long-Conv, Exp Init	19.9

Table 3: LRA Speed Benchmark.

Model	Speedup
Transformer	1×
FlashAttention	2.4×
Block-Sparse FlashAttention	2.8×
S4	2.9×
FLASHBUTTERFLY	**7.0**×

Table 4: Runtime and accuracy on Path256 (sequence length 64K).

Model	Accuracy	Training Time
Transformer	✗	✗
FLASHATTENTION	✗	✗
Block-Sparse FLASHATTENTION	63.1	3 days
FLASHBUTTERFLY	**92.2**	10 hours

classification accuracy than random guessing. Long convolutions are more robust to different initializations than variants of S4. However, regularization is critical for achieving strong performance; without it, long convolutions lose 17.2 points on average across the six LRA tasks.

Text Modeling: OpenWebText. We evaluate long convolutions as a drop-in replacement for SSMs in the H3 layer (Dao et al., 2022c), which stacks two SSMs and multiplies their outputs together as a gating mechanism. Following the H3 paper, we keep two attention layers in the overall language model and evaluate on OpenWebText. Table 2 shows the results. Long convolutions with random initialization come within 0.5 PPL points of H3, and the geometric decay initialization comes within 0.3 PPL. Both models outperform the Transformer. This initial result suggests that convolutions—with some multiplicative gating mechanism—may be a promising candidate to replace attention in language modeling.

4.2 EFFICIENCY: FLASHBUTTERFLY

Runtime on Long Range Arena. The Long Range Arena benchmark evaluates the efficiency of sequence models using runtime on a byte-level text classification benchmark. Table 3 compares a long convolution with FLASHBUTTERFLY against Transformers, and S4 with FLASHCONV. FLASHBUTTERFLY outperforms S4, since it does not require kernel generation, and outperforms Transformers by 7.0×.

Very Long Sequence Lengths. We demonstrate the utility of FLASHBUTTERFLY by training models on a task with extremely long sequences: Path256, which has sequence length 64K. Table 4 shows that long convolutions achieve state-of-the-art performance on Path256, outperforming block-sparse FLASHATTENTION from (Dao et al., 2022b), the only prior work to report non-trivial performance (>50% accuracy) on Path256. Long convolutions with FLASHBUTTERFLY exceed state-of-the-art performance by 29.1 points, and train 7.2× faster.

5 CONCLUSION

We show that long convolutions are a simple, yet effective approach to long sequence modeling. We find that regularizing the kernel weights with a squash operator allows long convolutions to achieve strong performance on a variety of long sequence modeling tasks. Finally, we develop FLASHBUTTERFLY to improve the runtime efficiency of long convolutions.

REFERENCES

Manohar Ayinala, Michael Brown, and Keshab K Parhi. Pipelined parallel fft architectures via folding transformation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20 (6):1068–1081, 2011.

Jun Ho Bahn, Jung Sook Yang, Wen-Hsiang Hu, and Nader Bagherzadeh. Parallel fft algorithms on network-on-chips. *Journal of Circuits, Systems, and Computers*, 18(02):255–269, 2009.

David H Bailey. FFTs in external or hierarchical memory. *The journal of Supercomputing*, 4(1):23–35, 1990.

Deanna M Barch, Gregory C Burgess, Michael P Harms, Steven E Petersen, Bradley L Schlaggar, Maurizio Corbetta, Matthew F Glasser, Sandra Curtiss, Sachin Dixit, Cindy Feldt, et al. Function in the human connectome: task-fmri and individual differences in behavior. *Neuroimage*, 80:169–189, 2013.

AJAA Bekele. Cooley-tukey fft algorithms. *Advanced algorithms*, 2016.

E Oran Brigham. *The fast Fourier transform and its applications*. Prentice-Hall, Inc., 1988.

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. In *International Conference on Learning Representations*, 2021.

Eleanor Chu and Alan George. *Inside the FFT black box: serial and parallel fast Fourier transform algorithms*. CRC press, 1999.

James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965. ISSN 00255718, 10886842. URL http://www.jstor.org/stable/2003354.

Kamalaker Dadi, Gaël Varoquaux, Antonia Machlouzarides-Shalit, Krzysztof J Gorgolewski, Demian Wassermann, Bertrand Thirion, and Arthur Mensch. Fine-grain atlases of functional modes for fmri analysis. *NeuroImage*, 221:117126, 2020.

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*, pp. 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*, pp. 4690–4721. PMLR, 2022a.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022b.

Tri Dao, Daniel Y Fu, Khaled K Saab, Armin W Thomas, Atri Rudra, and Christopher Ré. Hungry hungry hippos: Towards language modeling with state space models. *arXiv preprint arXiv:2212.14052*, 2022c.

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. In *International Conference on Learning Representations*, 2020.

Bruce Fischl. Freesurfer. *Neuroimage*, 62(2):774–781, 2012.

Karan Goel, Albert Gu, Chris Donahue, and Christopher Ré. It's raw! audio generation with state-space models. *arXiv preprint arXiv:2202.09729*, 2022.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *The International Conference on Learning Representations (ICLR)*, 2022a.

Albert Gu, Isys Johnson, Aman Timalsina, Atri Rudra, and Christopher Ré. How to train your hippo: State space models with generalized orthogonal basis projections. *arXiv preprint arXiv:2206.12037*, 2022b.

John Guibas, Morteza Mardani, Zongyi Li, Andrew Tao, Anima Anandkumar, and Bryan Catanzaro. Adaptive fourier neural operators: Efficient token mixers for transformers. *arXiv preprint arXiv:2111.13587*, 2021.

Ankit Gupta, Albert Gu, and Jonathan Berant. Diagonal state spaces are as effective as structured state spaces. In *Advances in Neural Information Processing Systems*, 2022.

Ramin Hasani, Mathias Lechner, Tsun-Huang Wang, Makram Chahine, Alexander Amini, and Daniela Rus. Liquid structural state-space models. *arXiv preprint arXiv:2209.12951*, 2022.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Maedbh King, Carlos R Hernandez-Castillo, Russell A Poldrack, Richard B Ivry, and Jörn Diedrichsen. Functional boundaries in the human cerebellum revealed by a multi-domain task battery. *Nature neuroscience*, 22(8):1371–1378, 2019.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. Fnet: Mixing tokens with fourier transforms. *arXiv preprint arXiv:2105.03824*, 2021.

Yuhong Li, Tianle Cai, Yi Zhang, Deming Chen, and Debadeepta Dey. What makes convolutional models great on long sequence modeling? *arXiv preprint arXiv:2210.09298*, 2022.

Youwei Liang, GE Chongjian, Zhan Tong, Yibing Song, Jue Wang, and Pengtao Xie. Evit: Expediting vision transformers via token reorganizations. In *International Conference on Learning Representations*, 2021.

Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. Mega: moving average equipped gated attention. *arXiv preprint arXiv:2209.10655*, 2022.

Harsh Mehta, Ankit Gupta, Ashok Cutkosky, and Behnam Neyshabur. Long range language modeling via gated state spaces. *arXiv preprint arXiv:2206.13947*, 2022.

Eric Nguyen, Karan Goel, Albert Gu, Gordon Downs, Preey Shah, Tri Dao, Stephen Baccus, and Christopher Ré. S4nd: Modeling images and videos as multidimensional signals with state spaces. In *Advances in Neural Information Processing Systems*, 2022.

NVIDIA. cufft v11.7.1 documentation, 2022. https://docs.nvidia.com/cuda/cufft/index.html.

Alan V Oppenheim. Applications of digital signal processing. *Englewood Cliffs*, 1978.

Alan V Oppenheim, John R Buck, and Ronald W Schafer. *Discrete-time signal processing. Vol. 2*. Upper Saddle River, NJ: Prentice Hall, 2001.

D.S. Parker. *Random Butterfly Transformations with Applications in Computational Linear Algebra*. CSD (Series). UCLA Computer Science Department, 1995.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, Chloe Hillier, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2019.

David W Romero, Anna Kuzina, Erik J Bekkers, Jakub Mikolaj Tomczak, and Mark Hoogendoorn. Ckconv: Continuous kernel convolution for sequential data. In *International Conference on Learning Representations*, 2021.

Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*, 2022.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2020.

Armin W Thomas, Christopher Ré, and Russell A Poldrack. Self-supervised learning of brain dynamics from broad neuroimaging data. *arXiv preprint arXiv:2206.11417*, 2022.

Asher Trockman and J Zico Kolter. Patches are all you need? *arXiv preprint arXiv:2201.09792*, 2022.

Gül Varol, Ivan Laptev, and Cordelia Schmid. Long-term temporal convolutions for action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 40(6):1510–1517, 2017.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 11106–11115, 2021.

A RELATED WORK

State space models Following S4 (Gu et al., 2022a), several SSM-based deep learning models have been proposed to model long sequences in different domains. In computer vision, S4ND (Nguyen et al., 2022) modifies the S4 architecture to work on 2D data. Goel et al. observe that S4 is unstable during autoregressive generation and propose a new architecture called SaShiMi to resolve this issue. SaShiMi draws a connection to Hurwitz matrices, allowing it to set state-of-the-art on unconditional waveform generation in the autoregressive setting. S5 was introduced as a multi-input, multi-output extension of S4 (Smith et al., 2022). Mehta et al. introduce a layer named Gated State Space (GSS) to effectively do autoregressive sequence modeling by using a gated state space architecture. Finally, there have been developments of gated attention-based mechanisms for long sequence modeling (Ma et al., 2022).

Convolutions In computer vision, convolutional models have set state-of-the-art on many tasks (He et al., 2016; Krizhevsky et al., 2017). However, these models are typically based on short, localized convolutions. Recently, there has been growing interest in developing models that use long, global convolutions. For example, models using long convolutions over the time dimension have been used to learn video representations (Varol et al., 2017). SGConv proposes a long convolution kernel to model long sequences. This model is based on two guiding principles: decay in the sequence dimension and a sublinear parameter count (Li et al., 2022). There has also been work formulating the convolution as a continuous function (Romero et al., 2021).

Transformer based architectures There has been recent progress using transformers to model long sequences, despite quadratic scaling in sequence length for attention-based models. In computer vision, several models have been proposed that build on the vision transformer (ViT) architecture (Dosovitskiy et al., 2020). For example, Trockman & Kolter introduce ConvMixer, which operates directly on patches of an image arranged in a sequence and uses a convolutional model to separately mix the channel and sequence dimensions. Liang et al. improve inference speed compared to ViT by reorganizing image tokens during the forward pass, fusing inattentive tokens. Other methods to use transformers for long sequence modeling include Rae et al., which presents the Compressive Transformer to enable long-range sequence modeling by compressing past memories.

Structured Matrices Several works have proposed to replace dense parameter matrices in neural networks with structured matrices (e.g., low-rank matrices, sparse matrices) in order to reduce network memory and compute requirements. One important line of work in structured matrices is based on butterfly matrices (Parker, 1995; Dao et al., 2019). Chen et al. aimed to improve the hardware-efficiency of butterfly matrices by using simple variants of butterfly. Dao et al. introduce Monarch matrices as a class of hardware efficient and expressive matrices. Further, the authors show that approximating a dense matrix with a Monarch matrix can be done analytically.

FFT Algorithms The computational feasibility of long convolutional models depends on the Fast Fourier Transform (FFT). The Cooley-Tukey FFT algorithm, published in 1965 (Cooley & Tukey, 1965), enabled convolution and Fourier transforms to scale in the length dimension from $O(N \log N)$ instead of $O(N^2)$. Subsequently, many alternative algorithms for efficiently computing the Fourier transform have emerged, including algorithms for computing the FFT in parallel (Ayinala et al., 2011). These algorithms have enabled fundamental progress in a range of disciplines, including control theory (Brigham, 1988; Bekele, 2016) and signal processing (Oppenheim, 1978; Oppenheim et al., 2001). A survey of methods is included in Chu & George; Bahn et al..

Finally, there has been work that trains neural networks in the Fourier domain, for example using token mixing (Guibas et al., 2021). FNet (Lee-Thorp et al., 2021) speeds up the transformer encoder architecture by using a Fourier transform to mix input tokens.

B ADDITIONAL EXPERIMENTS

B.1 IMAGE CLASSIFICATION

We evaluate long convolutions on image classification. We evaluate two settings which have been used to evaluate SSMs and sequence models: 1D pixel-by-pixel image classification, and 2D image classification. These settings are challenging for sequence modeling, as they require modeling complex spatial relationships between image pixels in a continuous space. For the 1D case, we use long

Table 5: Image classification on flattened images.

Model	sCIFAR
Transformer	62.2
LSTM	63.0
r-LSTM	72.2
UR-LSTM	71.0
UR-GRU	74.4
HIPPO-RNN	61.1
LipschitzRNN	64.2
CKConv	64.2
S4-LegS	91.8
S4-FouT	91.2
S4D-LegS	89.9
S4D-Inv	90.7
S4D-Lin	90.4
Long Conv, Random	91.4
Long Conv, Exp Init	**92.1**

Table 6: Image classification on 2D images.

Model	CIFAR
S4ND-ISO	**89.9**
Long Conv 2D-ISO, Rand init	88.1
Long Conv 2D-ISO, Exp init	89.1

Table 7: Evaluation on brain fMRI data.

Model	MAE
Transformer	0.68
H3	0.70
H3 + Long Convs, Rand Init	.58
H3 + Long Convs, Exp Init	**0.54**

Table 8: Univariate long sequence time-series forecasting results on ETTh1 Informer benchmark. Comparisons across five horizon prediction settings. Best mean squared error (MSE) and mean absolute error (MAE) in bold. Numbers reported from Gu et al. (2022a). Long Convs outperforms S4 and obtains best MSE and MAE in four out of five evaluation settings.

Methods	Long Convs		S4		Informer		LogTrans		Reformer		LSTMa		DeepAR		ARIMA		Prophet	
Metric	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE	MAE
24	**0.060**	0.202	0.061	**0.191**	0.098	0.247	0.103	0.259	0.222	0.389	0.114	0.272	0.107	0.28	0.108	0.284	0.115	0.275
48	**0.074**	**0.205**	0.079	0.22	0.158	0.319	0.167	0.328	0.284	0.445	0.193	0.358	0.162	0.327	0.175	0.424	0.168	0.33
168	**0.070**	**0.210**	0.104	0.258	0.183	0.346	0.207	0.375	1.522	1.191	0.236	0.392	0.239	0.422	0.396	0.504	1.224	0.763
336	0.082	**0.228**	**0.080**	0.229	0.222	0.387	0.23	0.398	1.86	1.124	0.59	0.698	0.445	0.552	0.468	0.593	1.549	1.82
720	**0.085**	**0.241**	0.116	0.271	0.269	0.435	0.273	0.463	2.112	1.436	0.683	0.768	0.658	0.707	0.659	0.766	2.735	3.253

convolutions as a drop-in replacement for the SSM layer in the state-of-the-art S4 architecture. For the 2D case, we replace the S4 layers in S4ND (Nguyen et al., 2022) with 2D long convolution filters.

Tables 5 and 6 show the results. On 1D image classification, long convolutions again match the performance of S4, even with random initializations, while their performance improves further by 1.3 points when using the exponential decay initialization. On 2D image classification, long convolutions come within 0.8 points of the state-of-the-art S4ND model—which suggests that higher dimensions may require different techniques or inductive bias to recover the same performance.

B.2 Time Series Forecasting

Time series forecasting is another challenging modality for sequence modeling, which requires reasoning over multiple time contexts. We evaluate the performance of long convolutions on different future horizon prediction windows in ETTh_1, a real-world long sequence time series forecasting task

Table 9: Downstream performance on brain fMRI data.

Dataset	Model	F1
MDTB	Transformer	91.8
	H3	92.0
	H3 + Long Convs, Rand Init	**92.1**
	H3 + Long Convs, Exp Init	91.6
HCP	Transformer	83.4
	H3	82.6
	H3 + Long Convs, Rand Init	82.3
	H3 + Long Convs, Exp Init	**83.6**

from the Informer benchmark Zhou et al. (2021). Following the original S4 paper, we evaluate on the univariate ETTh_1 task, which involves predicting electricity transformer temperature at hour-long granularities (i.e., 24, 48, 168, 336, and 720 hours in the future). For each prediction task, we use the same number of hours before as a look-back window to input to the model. As LongConvs can be a drop-in replacement for the S4 kernel, we also follow the approach taken in S4 that simply masks out the future time steps in the input sequence and treat the task as a masked sequence-to-sequence transformation. Table 8 shows the results. Long convolutions match or outperform S4 on all context windows, and outperforms custom hand-crafted architectures designed specifically for time series forecasting.

B.3 Brain fMRI Downstream Adaptation

We further evaluate the performance of the pre-trained models in two benchmark mental state decoding datasets from the Human Connectome Project (HCP; Barch et al., 2013) and multi-domain task battery (MDTB; King et al., 2019), spanning 20 and 26 distinct mental states respectively. To adapt the pre-trained models to the mental state decoding (i.e., classification) task, we add a learnable classification embedding $E^{cls} \in \mathbb{R}^n$ to the end of input sequences X and forward the model's corresponding prediction to a decoding head $p(\cdot)$, composed of a dense hidden layer with e model units (one for each embedding dimension, with $tanh$ activation) as well as a $softmax$ output layer (with one model unit i for each considered mental state in the data). Accordingly, we adapt models by optimizing a standard cross entropy loss objective: $-\sum_i y_i \log p(f(E^X))_i$, where y_i indicates a binary variable that is 1 if i is the correct mental state and 0 otherwise. We always begin downstream adaptation with the pre-trained model parameters and allow all parameters to change freely during training. We randomly split each of the two downstream datasets into distinct training (90% of fMRI runs) and test (10% of fMRI runs) datasets and adapt models for $1{,}000$ training steps at a mini-batch size of 256 and a learning rate of $5e^{-5}$ (otherwise using the same learning parameters as for upstream training). During training, we sample sequences from the fMRI datasets according to the accompanying event files, which specify the beginning and end of each experimental trial underlying a mental state (when accounting for the temporal delay of the haemodynamic response function; for details, see Thomas et al., 2022).

The adapted H3 variants with long convolutions perform on par with the other models in accurately identifying the mental states of the downstream evaluation datasets (see Table 9: F1-scores are macro-averaged).

C Methods Details

We discuss details of our methods.

C.1 Kernel Fusion

Naive implementations of the FFT convolution incur expensive GPU memory IO. Each FFT and inverse FFT operation requires at least one read and write of the input sequence from GPU memory, and so does the pointwise multiplication operation. For long sequences, the IO costs may be even worse: the entire input sequence cannot fit into SRAM, so optimized implementations such as cuFFT (NVIDIA, 2022) must take multiple passes over the input sequence using the Cooley-Tukey decomposition of the FFT (Cooley & Tukey, 1965). Following FlashAttention (Dao et al., 2022b), FlashButterfly's first contribution is to fuse the entire FFT convolution into a single kernel and compute the result directly in GPU SRAM to avoid this overhead.

Table 10: Runtime, GLOPs, and FLOP util for the Butterfly decomposition with different block sizes r for sequence length 4096, on A100 with batch size 128, head dimension 32.

Block Size	Runtime (ms)	GLOPs	FLOP Util
2	0.52	2.0	1.3%
16	0.43	8.1	6.0%
64	0.53	21.5	13.0%
256	0.68	64.5	30.4%

C.2 BUTTERFLY DECOMPOSITION

Kernel fusion reduces the IO requirements, but the fused FFT operations still cannot take full advantage of specialized matrix multiply units on modern GPUs, such as Tensor Cores on Nvidia GPUs, which perform fast 16×16 matrix multiplication. We appeal to a classical result, also known as the four-step or six-step FFT algorithm (Bailey, 1990), that rewrites the FFT as a series of block-diagonal Butterfly matrices (Parker, 1995) interleaved with permutation.

The Butterfly decomposition states that we can decompose an N-point FFT into a series of FFTs of sizes N_1 and N_2, where $N = N_1 N_2$. Conceptually, the algorithm reshapes the input as an $N_1 \times N_2$ matrix, applies N_1 FFTs of size N_2 to the columns, multiplies each element by a twiddle factor, and then applies N_2 FFTs of size N_1 to the rows.

More precisely, let $\mathbf{F}_N$ denote the DFT matrix corresponding to taking the N-point FFT. Then, there exist permutation matrices $\mathbf{P}$, and a diagonal matrix $\mathbf{D}$, such that $\mathbf{F}_N = \mathbf{P}(\mathbf{I}_{N_2} \otimes \mathbf{F}_{N_1})\mathbf{P}^T \mathbf{D}(\mathbf{I}_{N_1} \otimes \mathbf{F}_{N_2})\mathbf{P}$. $\mathbf{P}$ denotes a permutation matrix that reshapes the input to $N_1 \times N_2$ and takes the transpose, $\mathbf{D}$ denotes a diagonal matrix with the twiddle factors along the diagonal, $\otimes$ denotes the Kronecker product, and vI_{N_i} and $\mathbf{F}_{N_i}$ are the identity and DFT matrices of size $N_i \times N_i$. Precise values for $\mathbf{F}_{N_i}$, $\mathbf{D}$, and $\mathbf{P}$ are given in Appendix C.

The Butterfly decomposition incurs $O(Nr\log N/\log r)$ FLOPS for a sequence length $N = r^p$, with *block size* r. In general FFT implementations, N is typically padded to a power of two, so that the block size can be set to 2 to minimize the total number of FLOPS. However, on GPUs with a specialized $b \times b$ matrix multiply unit, the FLOP cost of computing an $r \times r$ matrix multiply with $r < b$ is equivalent to performing a single $b \times b$ matrix multiply. Thus the actual FLOP count scales as $O(Nb\log N/\log r)$ for $r < b$. Increasing the block size up to b actually *reduces* the FLOP cost.

Table 10 demonstrates this tradeoff on an A100 GPU, which has specialized matrix multiply units up to 16×32. Runtime decreases as r increases from 2, even though theoretical FLOPS increase. Once $r > b$, runtime begins increasing as actual FLOPS increase as well. We describe how to construct $\mathbf{D}$ in the Butterfly decomposition, and $\mathbf{B}$ in the three pass algorithm.

Twiddle Matrices We describe how to construct $N_1 \times N_2$ Twiddle matrices.

Let $M \in \mathbb{C}^{N_1 \times N_2}$. Then $M_{j,k} = \exp(-2\pi ijk/N)$. The twiddle factors $\mathbf{D}$ can be constructed by flattening M and using them along the diagonal of $\mathbf{D}$.

Butterfly Matrix We construct $\mathbf{B}$ in the three pass algorithm.

Let $\mathbf{B}^{(m)}$ denote the Butterfly matrix that needs to be constructed for a three pass algorithm with $N = lm$, and assume that m is a power of 2. $\mathbf{B}^{(m)}$ is a block matrix, where each block is a diagonal matrix. In particular, we have:

$$\mathbf{B} = \begin{bmatrix} \mathbf{D}_{1,1} & \dots & \mathbf{D}_{1,m} \\ \vdots & \ddots & \vdots \\ \mathbf{D}_{m,1} & \dots & \mathbf{D}_{m,m}, \end{bmatrix}.$$

We show how to construct $\mathbf{D}_{j,k}$. $\mathbf{D}_{j,k}$ is a diagonal matrix of size $l \times l$. The entries of $\mathbf{D}_{j,k}$ are given by the following:

$$\mathbf{D}_{j,k}[\tau] = \exp(-2i\pi k(jl + \tau)/N).$$

Algorithm 1 FLASHBUTTERFLY

Require: Input $u \in \mathbb{R}^{B\times H\times N}$, $\mathbf{K} \in \mathbb{R}^{H\times N}$, $\mathbf{D} \in \mathbb{R}^{H}$, where $N = lm$ is the sequence length, H is the head dimension, and B is the batch size.
1: $\hat{\mathbf{K}} \leftarrow FFT(\mathbf{K})$
2: $\mathbf{D}'_{\mathbf{K}} \leftarrow \mathbf{P}(\hat{\mathbf{K}})\mathbf{P}^{-1}$
3: $u \leftarrow \overline{\mathbf{B}}^{-1} u$
4: Compute $u \leftarrow (\mathbf{I}_m \otimes \overline{\mathbf{F}}_l)\mathbf{D}'_{\mathbf{K}}(\mathbf{I}_m \otimes \mathbf{F}_l)u$ in parallel across m streaming multiprocessors
5: Return $\overline{\mathbf{B}}u \in \mathbb{R}^{B\times H\times N}$

C.3 Additional details about the three pass algorithm

We share a few additional details about the three pass algorithm that allow for efficient training.

The butterfly matrices $\mathbf{B}$ have complex coefficients. Typically, we train models over real time series. This mismatch has the potential to increase the amount of GPU memory IO: it is necessary to read N real numbers, but write N complex numbers.

We can alleviate this problem by using a well-known transformation between a real FFT of length $2L$ and a complex FFT of length L (Brigham, 1988). In essense, a real FFT of length $2L$ can be converted into a complex FFT of length L. In our algorithm, we exploit this as follows:

- Given an input of real points N, reshape the input to be a complex input of length $N/2$.
- Compute the complex FFT convolution over the input of length $N/2$ using the three pass algorithm.
- Convert the output to be a real output of length N.

The first and last steps can be fused with a Butterfly matrix multiplication kernel, thereby keeping the total IO cost the same as the original algorithm.

D Theory

D.1 Three-Pass Algorithm

The full algorithm for FLASHBUTTERFLY for $N > l$ is shown in Algorithm 1.

We show that Algorithm 1 is correct, and that it can be computed in three passes over the input sequence.

Proposition 1. *Algorithm 1 computes the convolution $u * \mathbf{K}$ with at most three passes over the input sequence u.*

We prove Proposition 1.

Convolution Recall that a convolution between two vectors u and k of length N is given by the following:

$$u * k = \bar{\mathbf{F}}_L \text{Diag}(\mathbf{F}_L k)\mathbf{F}_L u.$$

We can precompute $\bar{\mathbf{F}}_L k$, since it is shared across all inputs in a batch. Let $\mathbf{D} = \bar{\mathbf{F}}_L k$. Then, the above is given by:

$$u * k = \bar{\mathbf{F}}_L \mathbf{D}\mathbf{F}_L u.$$

Decomposition One property of $\mathbf{F}_L$ is that it can be decomposed. For example, if $L = 2l$, then we can write the following:

$$\mathbf{F}_{2l} = \mathbf{B}\begin{bmatrix} \mathbf{F}_l & 0 \\ 0 & \mathbf{F}_l \end{bmatrix}\mathbf{P},$$

where $\mathbf{P}$ is a permutation matrix (in this case, an even-odd permutation), and $\mathbf{B}$ is a Butterfly matrix.

We can leverage this to re-write a convolution of length $2l$. Let u and k be vectors of length $2l$. Then, we can write the following:

$$\begin{aligned}
u*k &= \bar{\mathbf{F}}_{2l}\mathbf{D}\mathbf{F}_{2l}u \\
&= \bar{\mathbf{F}}_{2l}\mathbf{D}\bar{\mathbf{F}}_{2l}^{-1}u \\
&= \bar{\mathbf{B}}\begin{bmatrix}\bar{\mathbf{F}}_l & 0\\ 0 & \bar{\mathbf{F}}_l\end{bmatrix}\mathbf{P}\mathbf{D}\mathbf{P}^{-1}\begin{bmatrix}\bar{\mathbf{F}}_l^{-1} & 0\\ 0 & \bar{\mathbf{F}}_l^{-1}\end{bmatrix}\bar{\mathbf{B}}^{-1}u \\
&= \bar{\mathbf{B}}\begin{bmatrix}\bar{\mathbf{F}}_l & 0\\ 0 & \bar{\mathbf{F}}_l\end{bmatrix}\mathbf{D}'\begin{bmatrix}\bar{\mathbf{F}}_l^{-1} & 0\\ 0 & \bar{\mathbf{F}}_l^{-1}\end{bmatrix}\bar{\mathbf{B}}^{-1}u,
\end{aligned}$$

for some diagonal matrix $\mathbf{D}'$. Note that the three terms in the middle can be computed in parallel.

This pattern extends to $L = 2^m l$, and yields 2^m parallelism in the product.

It remains to show that each of the Butterfly matrices can be computed with a single read/write over the input sequence.

Recall that the Butterfly matrices have the following form:

$$\mathbf{B} = \begin{bmatrix}\mathbf{D}_{1,1} & \dots & \mathbf{D}_{1,m}\\ \vdots & \ddots & \vdots \\ \mathbf{D}_{m,1} & \dots & \mathbf{D}_{m,m},\end{bmatrix}$$

where the $\mathbf{D}_{i,j}$ are diagonal matrices of size $l \times l$.

A matrix-vector multiply $y = \mathbf{B}u$ can be partitioned on a GPU as follows. Suppose that each SM has enough shared memory to store l elements of the input. Let there be m SMs processing this input. Each SM will read l input and write l output, for $ml = N$ total reads and writes.

Specifically, SM i will read

$$\begin{aligned}
&u[(l/m)i:(l/m)(i+1)],\\
&u[l+(l/m)i:l+(l/m)(i+1)],\dots,\\
&u[(m-1)l+(l/m)i:(m-1)l+(l/m)(i+1)].
\end{aligned}$$

These inputs are exactly the inputs needed to compute:

$$\begin{aligned}
&y[(l/m)i:(l/m)(i+1)],\\
&y[l+(l/m)i:l+(l/m)(i+1)],\dots,\\
&y[(m-1)l+(l/m)i:(m-1)l+(l/m)(i+1)].
\end{aligned}$$

The SM can then distribute these portions of the matrix-vector multiply to the independent threads of the SM.

This completes the proof.

D.2 Expressivity of Long Convolutions

We show that long convolutions and SSMs are equivalent in expressivity (the subset relation in Figure 1 right is actually set equality).

Proposition 2. *Let M be a positive integer that evenly divides N. Any convolution kernel of length N can be written as the sum of N/M diagonal SSMs with hidden state M.*

Proof. For the case $M = 1$, consider a diagonal SSM with $\mathbf{A} \in \mathbb{R}^{N\times N}$ diagonal with entries $a_1,\dots,a_N$, and $\mathbf{B} \in \mathbb{R}^{N\times 1}$. For simplicity, we will roll $\mathbf{C}$ into $\mathbf{B}$ and set $\mathbf{D} = 0$.

This SSM gives rise to the following kernel $\mathbf{K}$ with entries:

$$\mathbf{K}_i = A^i\mathbf{B} = \sum_{j=1}^{N} a_j^i b_j.$$

This is equivalent to

$$\mathbf{K} = \mathbf{V}\mathbf{B},$$

where $\mathbf{V}$ is the transpose of a Vandermonde matrix

$$\mathbf{V}=\begin{bmatrix}1 & a_1 & a_1^2 & \dots & a_1^{N-1}\\ 1 & a_2 & a_2^2 & \dots & a_2^{N-1}\\ \dots & \dots & \dots & \dots & \dots\\ 1 & a_N & a_N^2 & \dots & a_N^{N-1}\end{bmatrix}^T.$$

Vandermonde matrices have a determinant that is nonzero if and only if $a_1,\ldots,a_N$ are all distinct. Thus $\mathbf{V}^T$ is invertible if $a_1,\ldots,a_N$ are distinct and hence $\mathbf{V}$ is also invertible if $a_1,\ldots,a_N$ are distinct. Given a kernel $\hat{\mathbf{K}}$, we can thus express that kernel by picking any $a_1,\ldots,a_N$ that are distinct and then picking $\mathbf{B}=\mathbf{V}^{-1}\hat{\mathbf{K}}$, then $\mathbf{V}\mathbf{B}=\mathbf{V}\mathbf{V}^{-1}\hat{\mathbf{K}}=\hat{\mathbf{K}}$, finishing the proof.

In the case where $M>1$ we have consider a diagonal SSM with $\mathbf{A}\in\mathbb{R}^{N\times N}$ diagonal with entries $a_1,\ldots,a_N$, and $\mathbf{B}\in\mathbb{R}^{N\times 1}$. Partition, the state N into N/M partitions of size M. Let $\sigma(i,j)$ denote the partition function that bijectively maps (i,j) pairs to $[1,\ldots,N]$ for $1\leq i\leq N/M, 1\leq j\leq M$.

Then the convolution kernel has the following entries $\mathbf{K}_i$:

$$\mathbf{K}_l=\sum_{i=1}^{N}a_i^l b_i=\sum_{i=1}^{N/M}\sum_{j=1}^{M}a_{\sigma(i,j)}^l b_{\sigma(i,j)}.$$

Consider the inner sum $\sum_{j=1}^{M}a_{\sigma(i,j)}^l b_{\sigma(i,j)}$. This defines a convolution kernel given by a diagonal SSM with hidden state M, $\mathbf{A}$ with diagonal entries $[a_{\sigma(i,1)},\ldots,a_{\sigma(i,M)}]$, and $\mathbf{B}=[b_{\sigma(i,1)},\ldots,b_{\sigma(i,M)}]^T$.

Thus, this diagonal SSM with hidden state N is the sum of N/M diagonal SSMs with hidden state M. □

Proposition 2 suggests that long convolutions and SSMs have fundamentally the same expressive power, especially when SSMs are used in a deep architecture that stacks multiple independent SSMs in layers. The significance of this result is that this allows us to view SSMs and general long convolutions as the same construct.

E Experiment Details

We discuss all the details of our experiments.

Table 11: The values of the best hyperparameters found; LRA, images, language, and time series, and brain fMRI. LR is learning rate and WD is weight decay. BN and LN refer to Batch Normalization and Layer Normalization. We use random weight initialization in all runs.

	Depth	Features H	Norm	kernel LR	Dropout	λ	Batch Size	WD	Epochs	LR
ListOps	8	128	BN	0.0005	0.2	0.002	50	0.05	40	0.01
Text (IMDB)	6	256	BN	0.001	0.2	0.003	16	0.05	32	0.01
Retrieval (AAN)	6	256	BN	0.0001	0.1	0.004	32	0.05	20	0.01
Image	6	512	LN	0.001	0.2	0.003	25	0.05	200	0.01
Pathfinder	6	256	BN	0.001	0.3	0.001	64	0.03	200	0.004
Path-X	6	256	BN	0.0005	0.3	0.001	4	0.05	50	0.0005
sCIFAR	6	512	LN	0.001	0.2	0.001	50	0.05	300	0.01
2D CIFAR	4	128	LN	0.001	0	0.001	50	0.01	100	0.01
OpenWebText	12	768	LN	0.001	0	0.001	32	0.1	100B tokens	0.0003
Time Series	3	128	BN	0.001	0.2	0.003	50	0.01	50	1e-5
Brain Upstream	4	768	LN	0.001	0.2	0.0005	512	0.1	5000 steps	0.01
Brain Downstream	4	768	LN	0.001	0.2	0.00005	256	0.1	1000 steps	0.01

Hyperparameter Sweeps For all methods, we swept the following parameters:

- Kernel Dropout: [0.1, 0.2, 0.3, 0.4, 0.5]
- Kernel LR: [0.0001, 0.0005, 0.001]
- λ: [0.001, 0.002, 0.003, 0.004, 0.005]

Table 12: Convolution- and SSM-specific hyperparameters.

Model	Hyperparameters	Initializations
SSM	d, lr_A, lr_B, lr_C dropout, discretization	LegS, FouT, LegS/FouT Inv, Lin
Long Convs	λ, kernel LR, k, dropout	Random, Geometric

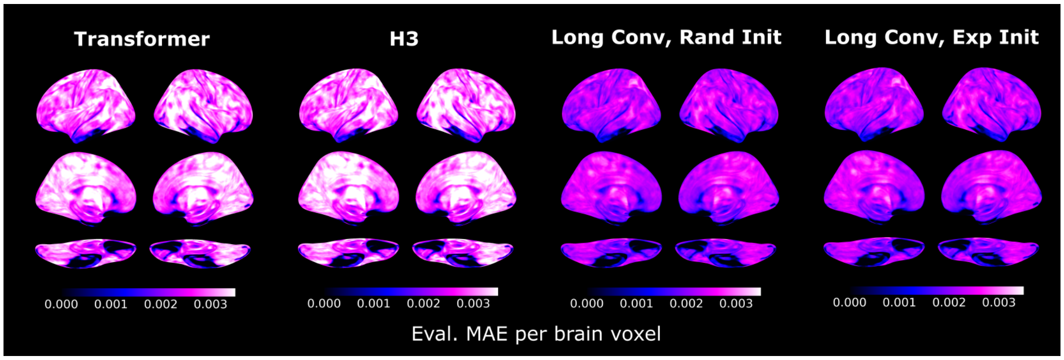

Figure 2: Mean absolute error of pre-trained models in upstream evaluation data for each location of the brain. Brain maps are projected onto the inflated cortical surface of the FsAverage template (Fischl, 2012).

Compute Infrastructure The experiments in this paper were run on a mixture of different compute platforms. The LRA experiments, except for Path-X, were swept on a heterogeneous cluster of 1xV100 and 2xV100 nodes. Path-X and sequential CIFAR were run on single 8xA100 nodes. The language modeling experiments were run on a single 8xA100 node. The time series experiments were run on a cluster with 1xP100 nodes. The brain fMRI experiments were run on a cluster of 2xV100 nodes.

Final Hyperparameters Final hyperparameters for reported results are given in Table 11.

Hyperparameter comparision to S4 Compared to the hyperparameters necessary to train S4, our regularization approaches have significantly fewer hyperparameters and choices than S4. Convolution-specific hyperparameters for S4 and long convolutions are shown in Table 12.

E.1 Functional Magnetic Resonance Imaging Data

Neuroimaging research can be considered as recently entering a big data era, as individual researchers publicly share their collected datasets more frequently. This development opens up new opportunities for pre-training at scale in neuroimaging research, as recently demonstrated by Thomas et al. (2022). In their work, the authors show that Transformers, pre-trained to predict brain activity for the next time point of input fMRI sequences, outperform other models in learning to identify the mental states (e.g., happiness or fear) underlying new fMRI data. Recently, Dao et al. (2022c) have shown that H3 performs on par with Transformers in this transfer learning paradigm.

To test whether long convolutions also perform on par with SSMs, as implemented in H3, and Transformers in this paradigm, we replicate the analyses of Thomas et al. (2022), using their published fMRI datasets. Conventionally, functional Magnetic Resonance Imaging (fMRI) data are represented in four dimensions, describing the measured blood-oxygen-level-dependent (BOLD) signal as a sequence $S=\{V_1,...,V_t\}$ of 3-dimensional volumes $V\in\mathbb{R}^{x\times y\times z}$, which show the BOLD signal for each spatial location of the brain (as indicated by the three spatial dimensions x, y, and z). Yet, due to the strong spatial spatial correlation of brain activity, fMRI data can also be represented differently, by representing individual sequences as a set $\Theta\in\theta_1,...,\theta_n$ of n functionally-independent brain networks θ, where each network describes the BOLD signal for some subset of voxels $v_{x,y,z}\in V$ (e.g., Dadi et al., 2020). The resulting sequences $X\in\mathbb{R}^{t\times n}$ indicate whole-brain activity as a set of n brain networks for t time points [1].

[1] Thomas et al. (2022) use $n=1{,}024$ networks defined by the Dictionaries of Functional Modes (DiFuMo; Dadi et al., 2020) Atlas.

Upstream learning: In line with Thomas et al. (2022), we pre-train models $f(\cdot)$ to predict whole-brain activity for the next time point j of an fMRI sequence X, using a mean absolute error (MAE) training objective, given the model's prediction $\hat{X} \in \mathbb{R}^{t \times n}$: MAE $= \frac{1}{n}\sum_{i=1}^{n} |X_{j,i} - \hat{X}_{j,i}|$; $\hat{X}_{t,n} = b_n + \sum_n f(E^X)_{t,e} w_{e,n}$; $E^X_{t,e} = E^{TR} + E^{pos} + b_e + \sum_n X_{t,n} w_{n,e}$. Here, $E^{TR} \in \mathbb{R}^e$ and $E^{pos} \in \mathbb{R}^e$ represent learnable embeddings for each possible time point and position of an input sequence (for details, see Thomas et al., 2022)[2]. Note that $f(\cdot)$ processes the input in a lower-dimensional representation $E^X \in \mathbb{R}^{t \times e}$, where $e = 768$, obtained through linear projection.

In line with Thomas et al. (2022) and Dao et al. (2022b), we pre-train a Transformer decoder (based on GPT) with 4 hidden layers and 12 attention heads and a H3 model with 4 hidden layers (with $H = 64$ and $m = 1$; see Dao et al., 2022c) in this task. For both models, the sequence of hidden-states outputs of the last model layer are used to determine $\hat{X}$ (scaled to the original input dimension with linear projection). We also pre-train variants of H3 that replace its SSM kernel with long convolutions.

We randomly divide the upstream data, which spans fMRI data from $11{,}980$ experimental runs of $1{,}726$ individuals, into distinct training and validation datasets by randomly designating 5% of the fMRI runs as validation data and using the rest of the runs for training. During training, we randomly sample sequences of 100 time points from the fMRI runs and train models with the ADAM optimizer (with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1e^{-8}$) for 5,000 steps at a mini-batch size of 512 and a learning rate of $5e^{-4}$. We also apply a linear learning rate decay schedule (with a warm-up phase of 10% of the total number of training steps), gradient norm clipping at 1.0, $L2$-regularisation (weighted by 0.1), and dropout at a rate of 0.2 (throughout all models). The adapted H3 variants clearly outperform the other models in accurately predicting brain activity for the next time point of input sequences (Table 7). We also find that the pre-trained models exhibit similar evaluation MAE error distributions throughout the brain, with relatively higher errors in the posterior parietal, occipital, and cingulate cortices as well parts of the limbic system (Fig. 2).

[2]As the sampling frequency of fMRI is variable between datasets, the same position of an input sequence can correspond to different time points.