BLOCKMAMBA: EFFICIENT SCALABLE STRUCTURED SPARSITY FOR MAMBA

Harshvardhan Mestha
BITS Pilani, K. K. Birla Goa Campus
Goa, India
f20220609@goa.bits-pilani.ac.in

Khaleelulla Khan Nazeer
Technische Universität Dresden, Germany
ScaDS.AI Dresden/Leipzig
khaleelulla.khan@tu-dresden.de

David Kappel
CITEC, Bielefeld University
Bielefeld, Germany
david.kappel@uni-bielefeld.de

Anand Subramoney
Royal Holloway, University of London
London, United Kingdom
anand.subramoney@rhul.ac.uk

ABSTRACT

State Space Models, particularly Mamba, have shown remarkable capabilities for efficiently scalable language modeling. To improve their efficiency for training and inference, we study the effect of introducing parameter sparsity and quantization in Mamba. Specifically, we introduce BlockMamba, a simple method to efficiently train sparse Mamba language models using block diagonal sparsity for the MLPs. Additionally, we also explore the effect of Power-of-Two quantization on model performance. Our results show that BlockMamba at 60-75% sparsity trains 35-40% faster than equivalently sized dense Mamba, with minimal loss in language modeling performance on GPUs. We also demonstrate that our method yields 15-20% inference speedup compared to dense on GPUs. We scale our approach by sparsifying a 1.4B parameter model, and demonstrate competitive language modeling performance and strong length extrapolation capability. We chose this combination of block sparsity and power-of-two quantization because they are also specifically advantageous for efficiency on alternative hardware such as neuromorphic or edge hardware. We provide a brief analysis with regard to efficiency gains on a neuromorphic hardware platform - the SpiNNaker2 system, to highlight BlockMamba's potential. Due to the low power characteristics of neuromorphic hardware, they can serve as mobile computing platforms for robotics and automotive applications, which makes them a useful target platform for such world models.

1 INTRODUCTION

With the introduction of the Transformer (Vaswani et al., 2023), Deep Learning models have demonstrated efficacy in a wide range of sequence modeling tasks, across modalities, mainly language modeling (Anthropic, 2024; Touvron et al., 2023). Transformers have proven effective at scale; however, their quadratic scaling with sequence length makes them computationally expensive. Developments such as KV-caching (Touvron et al., 2023), Sliding Windows (Beltagy et al., 2020), Attention Free transformers (Zhai et al., 2021), and hardware-aware algorithms such as FlashAttention (Dao et al., 2022) have mitigated their expense, but compromise in memory, context length limits.

State Space Models (Gu et al., 2022) address this by modeling sequences as a state space, and have linear scaling with sequence length. They demonstrated performance superior to transformers, especially for long context tasks. While the initial SSMs struggled with language modeling, later ones such as Mamba (Gu & Dao, 2024) were able to match the performance of transformers. Mamba achieves this by using a selection mechanism that provides the SSM with a global view of the sequence. Additionally, their hardware-aware algorithm leverages CUDA-enabled GPUs for efficient

computation, allowing them to scale on GPUs well. This has led to many works on Mamba-based LLMs (Zuo et al., 2024; Mistral AI team, 2024).

However, this dependency on CUDA kernel optimizations has made it challenging to port Mamba to other hardware platforms while retaining its performance. Hardware platforms such as neuromorphic hardware (Gonzalez et al., 2023) tend to benefit a lot from unstructured sparsity and different implementation paradigms compared to GPUs. Work has been done on making the SSM sparse in works such as (Meyer et al., 2024; Song et al., 2025; Tuo & Wang, 2025; Yang et al., 2024; Kim et al., 2025; Shihab et al., 2025). These methods focus on making the SSM parameters sparse using unstructured pruning, which accelerates Mamba effectively on CUDA GPUs. Current works on accelerating Mamba on FPGAs (Wang et al., 2025a) mainly focus on post-training quantization.

In this work, we study the impact of parameter sparsity and quantization in Mamba, for their applications both on GPUs and on alternative hardware. We introduce BlockMamba, a method to efficiently train Mamba models on CUDA and can lead to accelerated inference on various hardware. We specifically focus on the SpiNNaker2 platform as our target alternate hardware platform (Gonzalez et al., 2023) in addition to GPUs. We leverage structured block diagonal sparsity for the MLPs, as a drop-in replacement for the dense layers. Block diagonal sparsity speeds up training on GPUs and allows us to fit a larger SSM on SpiNNaker2. We demonstrate that our approach is faster during training on GPUs and is interoperable with existing Mamba weights, enabling us to finetune and merge models in the future.

Our contributions are as follows:

- We demonstrate that BlockMamba achieves 60-75% sparsity and trains 35-40% faster than equivalently sized dense Mamba, with minimal loss in language modeling performance on GPUs.
- We demonstrate that our approach works at scale, with minimal impact on length extrapolation capabilities.
- We also provide a brief analysis on how the design of BlockMamba yields significantly higher inference speedup on neuromorphic hardware, specifically SpiNNaker2.

2 METHOD

We use Mamba SSM (Gu & Dao, 2024) as our backbone, apply block diagonal structured sparsity and quantize the models with Power of Two quantization. These design choices are primarily motivated by the SpiNNaker2 neuromorphic chip's hardware topology, which enables faster inference, the implications of which are discussed in Section 3. We specifically opt for block diagonal sparsity only, as our aim is to increase parallelization on SpiNNaker2, which favors chunked tensor operations. Traditional methods, such as M:N sparsity (Zhou et al., 2021), are specifically optimized for Nvidia GPUs, which negates any benefit for SpiNNaker2.

But we show that we achieve speedups on GPUs as well using these methods. Our main focus is on the input and output dense projections in the Mamba layer, which surround the SSM. This allows us to easily port our method to other SSMs, and ensure that existing pretrained SSM weights can be loaded for finetuning and merging with BlockMamba.

2.1 BLOCK DIAGONAL SPARSITY

We employ Block Diagonal Linear layers for the input and output projections to introduce structured sparsity. Block diagonal sparsity restricts the weight matrix to a set of smaller, independent matrices along the main diagonal, while all off-diagonal elements are fixed to zero. We adopt the implementation from (Fu et al., 2023), which provides memory-optimized block diagonal matrix multiplication and supports mixed precision. A key modification we introduce to the standard Mamba architecture is the decoupling of the output projection from the SSM kernel. While the standard implementation fuses these operations for efficiency, maintaining this fusion with block diagonal layers would necessitate reconstructing a dense intermediate tensor to be compatible with the kernel, introducing a significant overhead. By separating them, we avoid this overhead. In this scheme, weight tensors are partitioned into n diagonal blocks. For instance, a weight matrix $W \in \mathbb{R}^{1024\times1024}$ with 4 blocks

becomes $W \in \mathbb{R}^{4\times256\times256}$, and with 8 blocks becomes $W \in \mathbb{R}^{8\times128\times128}$. We refer to models using this configuration as BlockMamba$_n$, where n denotes the number of blocks.

2.2 Power of Two quantization

We also apply quantizations to the input and output projections. We adapt the method from PoT-PTQ (Wang et al., 2025b), for the block diagonal sparse projection matrices, to use 3-dimensional tensors. We initialize the scales in accordance with their method, and do not perform calibration. We apply channel-wise quantization on the weights. Precisely we apply $\tilde{W}_{ij} = S_{ij} \cdot 2^{E_{ij}}$. Here, the exponent E is a signed 8-bit number, and S is a scale, initialized using a vectorized grid search to minimize error.

3 Analysis for neuromorphic hardware

3.1 The SpiNNaker2 System

SpiNNaker2 is an accelerator for large-scale event-based and asynchronous processing (Gonzalez et al., 2023). The SpiNNaker2 chip consists of 152 processing elements (PEs) connected via a network-on-chip (NoC). Each processing element is composed of an Arm M4f core, $128\,\mathrm{kB}$ SRAM, and a set of accelerators for exponential functions, random number generation, and multiply-accumulate (MAC) operations. The total of $19\,\mathrm{MB}$ on-chip SRAM is accompanied by $2\,\mathrm{GB}$ DRAM memory. Communication between the PEs in a single chip can be implemented by direct memory access (DMA) to other PEs' local memory. The local SRAM is organized into 4 memory banks of $32\,\mathrm{kB}$ each. One memory bank is usually reserved for program memory, and three banks for values such as weights and intermediate variables.

Each PE provides a Machine Learning Accelerator (MLA) for 8bit/16bit signed/unsigned execution of 2D matrix multiplication and 2D convolution (Zeinolabedin et al., 2022; Kelber et al., 2020). The accelerator consists of a 16x4 output-stationary multiply-and-accumulate (MAC) array and post-processing modules. The output can be quantized via shift and truncation in a post-processing block and written out as 8bit, 16bit or 32bit.

3.2 Effect of Block Diagonal Sparsity on SpiNNaker2 Inference

Due to the fine-grained 16x4 MAC array, the MLA can take advantage of the sparsity introduced by Block diagonal Linear projections. The speedup gained is proportional to the weights' sparsity.

Baseline case: In the baseline scenario, the weight matrix is dense, with no sparsity. The projection weight matrix of a single layer in Mamba 370M model is $W \in \mathbb{R}^{2048\times1024}$ is distributed across 32 PEs, each PE handling a submatrix of size 64×1024. The Matmul operations executed on each PE involve 64×1024 MAC operations and take approximately $350\mu s$ including quantization, preprocessing, and MLA access.

BlockMamba$_4$: Weight matrices use 4 diagonal blocks, hence the sparsity is $\frac{16-4}{16} = 0.75$. Each PE now handles a smaller submatrix $W_{PE} \in \mathbb{R}^{64\times256}$, reducing the number of MAC operations by 4. This results in the matmul time also being reduced by the same factor, taking approximately $88\mu s$ for computing the projection. Table 1 shows a comparison between the dense and BlockMamba$_4$ projection layers without the quantization.

BlockMamba$_8$: With 8 blocks, sparsity increases to 87.5%. Each PE processes a submatrix $W_{PE} \in \mathbb{R}^{64\times128}$, resulting in a matmul time of $44\mu s$ (about $\frac{1}{8}$ th of baseline)

BlockMamba$_{16}$ and BlockMamba$_{32}$: As the number of blocks increases, sparsity grows, and computational workload per PE decreases proportionally.

Memory usage and Read/Write bandwidth: Block diagonal sparsity also reduces memory usage and input read times. Memory usage for weights and inputs decreases proportionally with the

number of blocks, as inputs corresponding to zero weights are ignored. For example, reading the full input from external DRAM takes about $345\mu s$ in the baseline case, but this time decreases proportionally with the number of blocks.

We implement the Mamba 130M and 370M dense models on SpiNNaker2 and measure the time required by the full model. To estimate the BlockMamba$_4$ timings, we replace the measured in/out projection layer timings from Table 1 within the dense model's timing calculations. The time required for the dense model is 120 ms compared to the BlockMamba$_4$ model which takes 100 ms, see Table 2. Although the projection layers are sped-up by 4-3x the overall speedup is limited to 1.2x beacuse of other bottlenecks that exists in the system such of communication between layers and computaion of SSM layers which cannot be accelerated. Speedup on Nvidia GPU is only possible using CUDA kernels, the Block Diagonal projections add overhead which leads to marginal gains in inference, which is given in Table 4

Model	N	In proj. (μs)	Speedup	Out proj. (μs)	Speedup
Mamba (130M)	1 (dense)	84	–	17	–
	4	21	4.0×	8	2.1×
	8	11	7.6×	6	2.8×
	16	5	16.8×	6	2.8×
	32	3	28.0×	5	3.4×
Mamba (370M)	1 (dense)	116	–	28	–
	4	29	4.0×	12	2.3×
	8	15	7.7×	9	3.1×
	16	8	14.5×	8	3.5×
	32	4	29.0×	7	4.0×

Table 1: Measured timings for a single BlockMamba$_N$ projection layer on SpiNNaker2, with speedup vs. dense (N=1).

Model	End-to-end Latency (ms)		Estimated Speedup
	Dense (Actual)	BlockMamba$_4$ (Estimated)	
Mamba (130M)	66	54	1.2
Mamba (370M)	124	102	1.2

Table 2: Estimated speedup for the full model using BlockMamba$_4$ projection layers. BlockMamba is estimated to give a 20% speedup in inference for Mamba on SpiNNaker 2. Timing for Dense is measured and for BlockMamba$_4$ is estimated combining Dense and projection layers from Table 1

3.3 Effect of Power of Two Quantization on SpiNNaker2 Inference

Power-of-two quantization is a specialized quantization technique that leverages the simplicity of bit-shift operations to scale outputs efficiently. This section explores its motivation and practical implications for implementing Mamba on SpiNNaker2.

The post-processing module in the MLA can efficiently scale outputs using SIMD (Single Instruction, Multiple Data) bit-shift operations. This is only feasible when the quantization scale is a power of two, as bit-shifting is mathematically equivalent to multiplying or dividing by powers of two. Unlike floating-point scaling, which requires more computational resources, bit-shift operations introduce practically no overhead.

To enable power-of-two quantization, the network must be retrained or fine-tuned to align with the constraints of this quantization scheme. This ensures that the model's weights and activations are appropriately adjusted to maintain accuracy while adhering to the power-of-two scaling requirement. Without retraining, the quantization process may introduce significant errors, particularly in sensitive layers of the network.

Scaling outputs using floating-point operations on an Arm core is computationally expensive, taking approximately 2x the time of a matrix multiplication operation on the MLA. In contrast, power-of-two quantization eliminates this bottleneck by replacing floating-point multiplications with simple bit-shift operations. Which is orders of magnitude faster and more efficient, especially when performed using the SIMD operations.

4 RESULTS

4.1 LANGUAGE MODELING

For language modeling, we train the models from scratch on the MiniPile dataset (Kaddour, 2023), a subset of the Pile (Gao et al., 2020), containing 2 billion tokens, and measure perplexity (PPL). All hyperparameters are identical and are provided in Appendix A. GFLOPs measured using the ptflops (Sovrasov, 2018-2024) library for an input of batch size 32 and sequence length of 1024 at 16-bit precision for inference. Mamba baselines are trained at 32-bit precision as standard. BlockMamba is trained at 16-bit mixed precision, as the block diagonal MLPs support mixed precision. We found that training baseline Mamba at 16-bit precision is unstable. The number in subscript denotes the number of blocks each projection is divided into, as described above.

Table 3, shows the training time speedup for various block sizes for BlockMamba as well as the perplexity achieved. It can be seen that BlockMamba trains upto 34-42% faster than the dense standard 130M model with larger block sizes at the cost of higher perplexity. In principle, BlockMamba could be trained on a equivalent number of additional tokens to achieve even lower perplexities while matching the training time of the denser models.

Model	Params	Val PPL	Test PPL	GFLOPs	Train time (m)	Speedup
Mamba*	130M	18.86 ± 0.52	18.38 ± 0.53	525.39	559.88 ± 0.48	1
Mamba*	370M	17.05 ± 0.55	16.61 ± 0.56	756.13	1537.18 ± 8.36	0.37
BlockMamba$_4$	145M	19.25 ± 0.08	18.73 ± 0.09	292.28	412.05 ± 0.92	1.34
BlockMamba$_8$	107M	20.98 ± 0.17	20.37 ± 0.17	214.97	402.23 ± 1.15	1.39
BlockMamba$_{16}$	88M	23.37 ± 0.04	22.73 ± 0.06	176.31	397.78 ± 0.67	1.41
BlockMamba$_{32}$	78M	26.32 ± 0.14	25.62 ± 0.15	156.98	394.88 ± 0.35	1.42

Table 3: Language Modeling Performance and training time on MiniPile, mean and standard deviation over 3 runs. The runs for BlockMamba were with mixed precision training. * Full precision models were used for the baselines, since using mixed precision caused the training to be unstable.

4.2 INFERENCE SPEED

We compare inference speed between sparse and dense models with a reference random input of batch size 32. All models are at 16-bit floating-point precision. Table 4 shows the comparison between the dense and sparse models on 1x A100 Nvidia GPU for token throughput, in both generate and prefill modes. BlockMamba is 15% faster than the original dense 370M model, though it is much slower than the 130M dense model. This is expected as the Block diagonal weights during inference on CUDA GPUs add overhead, as many calls to the unoptimized batch matrix multiplication primitive are called. This can be further optimized using a specialized CUDA kernel for block diagonal matrix multiplication.

4.3 POST TRAINING

We also measure the impact of block diagonal sparsity on post-training quantization. Perplexity measured on the validation set of MiniPile. We report performance on the WikiText 2 task to measure the impact of sparsity and quantization on task generalization.

Results reported in Table 5 indicate that Power of Two quantization impacts performance less on larger models, with significant performance loss for BlockMamba$_{16}$, and BlockMamba$_{32}$.

		Throughput (↑ tokens/sec)				
	Sequence length	64	128	256	512	1024
Generate	Mamba (130M)	1606.6	1359.3	741.9	368.5	188.5
	Mamba (370M)	715.7	532.6	289.0	144.4	73.3
	BlockMamba$_4$ (145M)	698.5	594.8	332.2	165.6	84.3
	BlockMamba$_8$ (107M)	665.7	606.4	336.3	167.7	85.8
	BlockMamba$_{16}$ (88M)	681.7	616.3	337.9	168.3	85.6
	BlockMamba$_{32}$ (78M)	680.2	621.3	338.3	169.3	85.7
Prefill	Mamba (130M)	102780	175083	190082	188656	193362
	Mamba (370M)	45825	68221	74104	73820	75243
	BlockMamba$_4$ (145M)	44877	76205	85170	84889	86586
	BlockMamba$_8$ (107M)	44373	77664	85983	86197	88097
	BlockMamba$_{16}$ (88M)	44103	78964	86389	86437	87606
	BlockMamba$_{32}$ (78M)	43336	79543	86608	86607	87980

Table 4: Inference throughput (tokens/sec) comparison of dense Mamba vs. sparse BlockMamba variants on CUDA GPU. Column headers (64, 128, 256, 512, 1024) denote the context sequence length in tokens. Higher throughput is better (↑).

Model	Params (M)	Quantization	PPL MiniPile	PPL WikiText
Mamba	130	-	18.43	66.98
Mamba	130	Uniform (W8A8)	18.46	67.05
Mamba	130	PoT (W8A16)	19.37	69.14
Mamba	370	-	16.77	59.11
Mamba	370	Uniform (W8A8)	16.80	59.15
Mamba	370	PoT (W8A16)	17.37	60.41
BlockMamba$_4$	145.02	-	19.29	71.35
BlockMamba$_4$	145.02	Uniform (W8A8)	19.35	71.66
BlockMamba$_4$	145.02	PoT (W8A16)	20.35	76.32
BlockMamba$_8$	107.28	-	20.85	79.32
BlockMamba$_8$	107.28	Uniform (W8A8)	21.07	80.38
BlockMamba$_8$	107.28	PoT (W8A16)	22.26	86.41
BlockMamba$_{16}$	88.40	-	23.41	92.05
BlockMamba$_{16}$	88.40	Uniform (W8A8)	24.76	98.28
BlockMamba$_{16}$	88.40	PoT (W8A16)	25.58	102.83
BlockMamba$_{32}$	78.96	-	26.42	109.08
BlockMamba$_{32}$	78.96	Uniform (W8A8)	36.58	150.37
BlockMamba$_{32}$	78.96	PoT (W8A16)	34.87	154.67

Table 5: Language Modeling Performance of Power of Two Quantized models. We show that the PPL is not affected by PoT Quantization.

4.4 SCALING UP

To test whether our approach scales effectively, we also train a model on the first 20 billion tokens of the SlimPajama dataset using our method applied to a 1.4-billion-parameter Mamba model. We train only BlockMamba; the remainder of the results in Table 6 are as reported in the Samba (Ren et al., 2025) paper. We train on a sequence length of 1024 while the other models were trained on a sequence length of 4096, putting BlockMamba at a slight disadvantage. Additional training details in Appendix A.

In Table 6, Samba denotes the model in (Ren et al., 2025), with the'-NoPE' suffix denoting a Samba model trained without positional encodings of the sequence. The Mega-S6 is the Mega architecture (Ma et al., 2023), adapted to use Mamba layers instead of Multi-dimensional Damped Exponential

Moving Average (MD-EMA) layers, and also applies positional encoding. The results indicate that Sparsification doesn't hurt context length generalization and beats Mega-S6.

Table 6 demonstrates that BlockMamba is able to generalize to longer sequence lengths well, even though it was trained on a much shorter sequence length of 1024 compared to the other models. We also test the impact of quantization on the sparse models, applying uniform W8A8 quantization and Power-of-Two quantization (Wang et al., 2025b), without calibration.

Architecture	Size	Layers	Validation Context Length		
			4096	8192	16384
Llama-2	438M	24	11.14	47.23	249.03
Mamba	432M	60	10.70	10.30	10.24
Mega-S6	422M	24	12.63	12.25	12.25
SAMBA-NoPE	421M	24	10.11	28.97	314.78
SAMBA	421M	24	10.06	9.65	9.57
BlockMamba$_4$1.4B	466M	48	11.79	12.07	13.81
BlockMamba$_4$1.4B-W8A8	466M	48	12.60	12.50	14.59
BlockMamba$_4$1.4B-PoT	466M	48	12.12	13.03	15.29

Table 6: Perplexity and Length Extrapolation on the validation set of SlimPajama. We show that our models extrapolate well to longer sequences, even though they are trained on a context length of 1024. All other models were trained on a context length of 4096.

5 CONCLUSION

In this work, we studied the effect of introducing block-diagonal sparsity on training and inference speed, and that of power-of-two quantization on inference speed on GPUs. The experiments in Section 3 demonstrate a training speedup of 35-40% on a GPU. Table 4 indicates that the sparse model is 15% faster during both prefill and generate, compared to dense. The speedup comparison highlights the fact that CUDA GPUs are unable to leverage block diagonal sparsity effectively, since the gains are marginal. The quantization results indicate that using Power of Two quantization degrades generalizability significantly compared to standard uniform quantization for smaller models. Table 6 clearly demonstrates that BlockMamba is scalable, obtaining comparable perplexity with various architectures, even beating Mega-S6, an architecture that applies sequence chunking and also uses positional encodings. Our method also retains the length extrapolation capabilities of Mamba, despite being trained on 4 times shorter sequences than other models; perplexities are stable for even 16 times the training sequence length. Power of Two quantization degrades perplexity less at scale, but impacts length generalization more. The results also indicate that pushing sparsity beyond 4 blocks yields diminishing returns.

We include an analysis of expected speedups on SpiNNaker2 hardware platform, and a target platform of interest for the methods we introduce. The estimated gains there are significantly higher, as applying the projection layers becomes exponentially faster with the number of blocks. The methods we introduce has the potential to provide significant speedups on SpiNNaker 2. But, interestingly, they also provide some advantages on GPUs even if not designed to do so.

BlockMamba allows for larger SSMs to be used with a similar total parameter count, allowing models such as these to capture larger sequences. One application of this architecture is envisioned to be world models (Savov et al., 2025), where a frozen Mamba backbone for encoding actions could be replaced with BlockMamba to speed up training with minimal performance degradation or enable efficient sparse fine-tuning at scale. Due to the low power characteristics of neuromorphic hardware, they serve as mobile computing platforms for robotics and automotive applications, which makes them a useful target platform for such world models.

6 LIMITATIONS AND FUTURE WORK

The main limitation is that our implementation is not optimized for CUDA; a further gain in speed can be achieved by using GPU kernels dedicated for block diagonal matrix multiplication and quan-

tization. We motivate our design of BlockMamba to minimize the overhead due to the linear projection, as detailed in the earlier sections. A practical implementation of BlockMamba on SpiNNaker2 is yet to be benchmarked. Future work can focus on applying BlockMamba for scaling various world models, especially those requiring a large SSM to encode modalities with dense information, where using a larger state size is important. Event data is extremely dense, with the current state of the art architectures (Schöne et al., 2024), having to limit the number of events the model can process in a batch.

ACKNOWLEDGEMENTS

This work was partly funded by the German Federal Ministry for Economic Affairs and Climate Action (BMWK) project ESCADE (01MN23004D). This work was partly funded by BMBF project EVENTS (16ME0733). The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer JUWELS at Jülich Supercomputing Centre (JSC). The authors gratefully acknowledge the computing time made available to them on the high-performance computer at the NHR Center of TU Dresden. This center is jointly supported by the Federal Ministry of Research, Technology and Space of Germany and the state governments participating in the NHR (www.nhr-verein.de/unsere-partner).

REFERENCES

Anthropic. Introducing the next generation of claude: Claude 3 model family, 2024. URL https://www.anthropic.com/news/claude-3-family. Accessed: 2024-10-21.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL https://arxiv.org/abs/2004.05150.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness, 2022. URL https://arxiv.org/abs/2205.14135.

Daniel Y. Fu, Simran Arora, Jessica Grogan, Isys Johnson, Sabri Eyuboglu, Armin W. Thomas, Benjamin Spector, Michael Poli, Atri Rudra, and Christopher Ré. Monarch mixer: A simple sub-quadratic gemm-based architecture, 2023. URL https://arxiv.org/abs/2310.12109.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020. URL https://arxiv.org/abs/2101.00027.

Hector Andres Gonzalez, Jiaxin Huang, Florian Kelber, Khaleelulla Khan Nazeer, Tim Hauke Langer, Chen Liu, Matthias Aleander Lohrmann, Amirhossein Rostami, Mark Schöne, Bernhard Vogginger, Timo Wunderlich, Yexin Yan, Mahmoud Akl, and Christian Mayr. SpiNNaker2: A large-scale neuromorphic system for event-based and asynchronous machine learning. In *First Workshop on Machine Learning with New Compute Paradigms*, 2023. URL https://openreview.net/forum?id=KAiPD1OwvF.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL https://arxiv.org/abs/2312.00752.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces, 2022. URL https://arxiv.org/abs/2111.00396.

Oskar Hallström, Said Taghadouini, Clément Thiriet, and Antoine Chaffin. Passing the torch: Training a mamba model for smooth handover, 2024. URL https://www.lighton.ai/lighton-blogs/passing-the-torch-training-a-mamba-model-for-smooth-handover.

Jean Kaddour. The minipile challenge for data-efficient language models, 2023. URL https://arxiv.org/abs/2304.08442.

Florian Kelber et al. Mapping Deep Neural Networks on SpiNNaker2. In *NICE 2020*, pp. 1–3, 2020. ISBN 978-1-4503-7718-8. doi: 10.1145/3381755.3381778. URL https://doi.org/10.1145/3381755.3381778.

Jiyong Kim, Jaeho Lee, Jiahao Lin, Alish Kanani, Miao Sun, Umit Y. Ogras, and Jaehyun Park. emamba: Efficient acceleration framework for mamba models in edge computing, 2025. URL https://arxiv.org/abs/2508.10370.

Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. Mega: Moving average equipped gated attention, 2023. URL https://arxiv.org/abs/2209.10655.

Svea Marie Meyer, Philipp Weidel, Philipp Plank, Leobardo Campos-Macias, Sumit Bam Shrestha, Philipp Stratmann, and Mathis Richter. A diagonal structured state space model on loihi 2 for efficient streaming sequence processing, 2024. URL https://arxiv.org/abs/2409.15022.

Mistral AI team. Codestral mamba. https://mistral.ai/news/codestral-mamba, July 2024. Accessed: 2025-12-15.

Liliang Ren, Yang Liu, Yadong Lu, Yelong Shen, Chen Liang, and Weizhu Chen. Samba: Simple hybrid state space models for efficient unlimited context language modeling, 2025. URL https://arxiv.org/abs/2406.07522.

Nedko Savov, Naser Kazemi, Deheng Zhang, Danda Pani Paudel, Xi Wang, and Luc Van Gool. Statespacediffuser: Bringing long context to diffusion world models, 2025. URL https://arxiv.org/abs/2505.22246.

Mark Schöne, Yash Bhisikar, Karan Bania, Khaleelulla Khan Nazeer, Christian Mayr, Anand Subramoney, and David Kappel. Stream: A universal state-space model for sparse geometric data, 2024. URL https://arxiv.org/abs/2411.12603.

Ibne Farabi Shihab, Sanjeda Akter, and Anuj Sharma. Efficient unstructured pruning of mamba state-space models for resource-constrained environments, 2025. URL https://arxiv.org/abs/2505.08299.

Woomin Song, Jihoon Tack, Sangwoo Mo, Seunghyuk Oh, and Jinwoo Shin. Sparsified state-space models are efficient highway networks, 2025. URL https://arxiv.org/abs/2505.20698.

Vladislav Sovrasov. ptflops: a flops counting tool for neural networks in pytorch framework, 2018-2024. URL https://github.com/sovrasov/flops-counter.pytorch.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Kaiwen Tuo and Huan Wang. Sparsessm: Efficient selective structured state space models can be pruned in one-shot, 2025. URL https://arxiv.org/abs/2506.09613.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Aotao Wang, Haikuo Shao, Shaobo Ma, and Zhongfeng Wang. Fastmamba: A high-speed and efficient mamba accelerator on fpga with accurate quantization, 2025a. URL https://arxiv.org/abs/2505.18975.

Xinyu Wang, Vahid Partovi Nia, Peng Lu, Jerry Huang, Xiao-Wen Chang, Boxing Chen, and Yufei Cui. Potptq: A two-step power-of-two post-training for llms, 2025b. URL https://arxiv.org/abs/2507.11959.

Xinyu Yang, Jixuan Leng, Geyang Guo, Jiawei Zhao, Ryumei Nakada, Linjun Zhang, Huaxiu Yao, and Beidi Chen. S^2ft: Efficient, scalable and generalizable llm fine-tuning by structured sparsity, 2024. URL https://arxiv.org/abs/2412.06289.

Seyed Mohammad Ali Zeinolabedin et al. A 16-channel fully configurable neural soc with 1.52 μw/ch signal acquisition, 2.79 μw/ch real-time spike classifier, and 1.79 tops/w deep neural network accelerator in 22 nm fdsoi. *IEEE TBioCAS*, 16(1):94–107, 2022. doi: 10.1109/TBCAS.2022.3142987.

Shuangfei Zhai, Walter Talbott, Nitish Srivastava, Chen Huang, Hanlin Goh, Ruixiang Zhang, and Josh Susskind. An attention free transformer, 2021. URL https://arxiv.org/abs/2105.14103.

Aojun Zhou, Yukun Ma, Junnan Zhu, Jianbo Liu, Zhijie Zhang, Kun Yuan, Wenxiu Sun, and Hongsheng Li. Learning n:m fine-grained structured sparse neural networks from scratch, 2021. URL https://arxiv.org/abs/2102.04010.

Jingwei Zuo, Maksim Velikanov, Dhia Eddine Rhaiem, Ilyas Chahed, Younes Belkada, Guillaume Kunsch, and Hakim Hacid. Falcon mamba: The first competitive attention-free 7b language model, 2024. URL https://arxiv.org/abs/2410.05355.

A Appendix

A.1 Hyperparameters

The hyperparameters for the experiments in section 3 are: Learning Rate: 0.001, Batch Size: 4, Gradient accumulation: 8, Scheduler: Linear decay with no warmup, Hardware: 4x NVIDIA A100, 1 epoch, 0.5 gradient decay.

The hyperparameters for the experiments in section 6 are: Learning Rate: 0.0008 for 85%, 0.0006 for the remaining epoch, Batch Size: 64, Gradient accumulation: 4, Scheduler: Warmup Stable Decay scheduler (Hallström et al., 2024), Hardware: 4x NVIDIA H100, 1 epoch, 0.5 gradient decay.