BLACKMAMBA: MIXTURE OF EXPERTS FOR STATE-SPACE MODELS

Quentin Anthony* **Yury Tokpanov*** **Paolo Glorioso*** **Beren Millidge***
Zyphra
{quentin, yury, paolo, beren}@zyphra.com

ABSTRACT

State-space models (SSMs) have recently demonstrated competitive performance to transformers at large-scale language modeling benchmarks while achieving linear time and memory complexity as a function of sequence length. Mamba, a recently released SSM model, shows impressive performance in both language modeling and long sequence processing tasks. Simultaneously, mixture-of-expert (MoE) models have shown remarkable performance while significantly reducing the compute and latency costs of inference at the expense of a larger memory footprint. In this paper, we present BlackMamba, a novel architecture that combines the Mamba SSM with MoE to obtain the benefits of both. We demonstrate that BlackMamba performs competitively against both Mamba and transformer baselines, and outperforms in inference and training FLOPs. We fully train and open-source 340M/1.5B and 630M/2.8B BlackMamba models on 300B tokens of a custom dataset. We show that BlackMamba inherits and combines both of the benefits of SSM and MoE architectures, combining linear-complexity generation from SSM with cheap and fast inference from MoE. We release all weights, checkpoints, and inference code open-source. [1]

1 INTRODUCTION

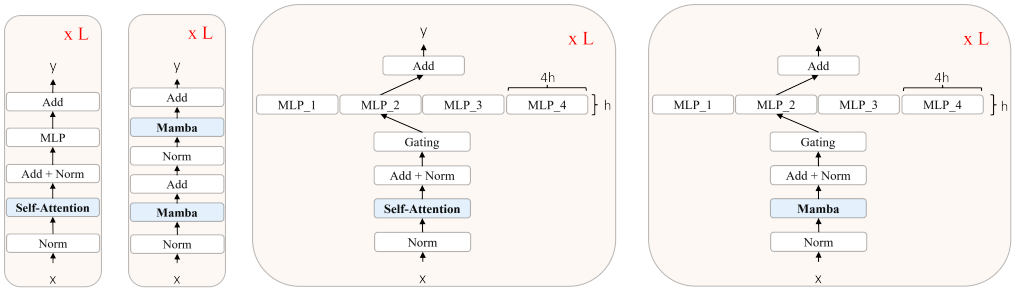
In order to ameliorate the computational demands of the attention mechanism, significant effort has recently been directed towards architectural alternatives to the canonical dense attention transformer model. Some of the most promising candidate architectures are State Space Models (SSMs) (1; 2) and Mixture of Experts (MoE) (3; 4; 5). The key practical benefit of SSMs over transformers is their linear computational complexity with respect to input sequence length (as opposed to the quadratic complexity of transformers). This theoretically enables SSMs to process vastly longer sequences than transformers for a given FLOP budget, and to render autoregressive generation constant in compute without a KV cache. Notable recent examples of SSMs include Mamba (1), RWKV (2), and RetNet (6), all of which demonstrate efficient long-sequence training and inference, efficient implementations in CUDA, and competitive language modeling task performance to transformers with similar scaling properties. At the same time mixture of expert (MoE) architectures (7; 8; 3; 4) have become an emerging advance over dense transformers which allow for significantly reduced training and inference FLOPs required to achieve comparable quality to a comparable dense model. MoE models allow for only a sparse subset of the total parameters to be activated on a single forward pass, relying on a routing function to gate which 'experts' are utilized or not depending on the context. This sparsity decouples the inference cost and parameter count of a model, enabling significantly stronger performance for a given inference budget at the cost of many more parameters and a correspondingly greater memory footprint.

These architectural improvements over transformers are compelling on their own, but we believe that their combination is a natural next step that could enable significantly improved language modelling speed and performance against the canonical transformer. Specifically, we expect a Mamba-MoE architecture would have the following improvements over a dense transformer: 1) *Mamba*: Linear

*All authors contributed equally to this work

[1] Inference code at: `https://github.com/Zyphra/BlackMamba`

computational complexity with respect to input sequence length for both training and inference. Autoregressive generation in constant time and memory. 2) *MoE*: Inference latency and training FLOPs of the equivalent smaller dense base model, while preserving model quality close to an equi-parameter dense model.



(a) Transformer (b) Mamba (c) Transformer-MoE (d) Mamba-MoE

Figure 1: Architecture of dense transformer, dense Mamba, transformer-MoE, and Mamba-MoE

In this paper, we begin to demonstrate that these improvements are achievable and that, when put together, these two approaches synergize to produce a model with compelling evaluation performance (Figs. 3-6), compute (Fig. 9), and latency advantages (Figs. 2(a) and 2(b)) over existing transformer models and which can be trained at a fraction of the FLOP cost for similar performance (Fig. 9). We study the MoE routing statistics exhibited by our model across training time and across model depth. Additionally, we introduce a novel initialization for our routing Sinkhorn algorithm which significantly reduces the number of iterations required until convergence, thus improving routing speed.

The main achievements of this work are:

- We design, implement, and evaluate **BlackMamba**: a combination of alternating attention-free Mamba blocks and routed MLPs.
- We train and open-source two BlackMamba Models[2]: BlackMamba 340M/1.5B and BlackMamba 630M/2.8B.
- We demonstrate that BlackMamba requires significantly fewer training FLOPs to achieve comparable downstream task performance to a dense transformer model.
- We explore the compounding inference benefits of the combination of attention-free architectures such as Mamba along with routed sparsity architectures such as MoE.

The final checkpoints are open-sourced on HuggingFace with Apache 2.0 licensing, and intermediate training checkpoints are available upon request. Inference code is provided at `https://github.com/Zyphra/BlackMamba`.

2 Related Work

A number of recent works (6; 9) has aimed to increase the expressivity of the state-space model by using input-dependent gating, similar to the QKV matrices of attention, while maintaining the fundamentally linear nature of the state-space recursion. Mamba (1) is a recently released state-space model in line with these previous works which demonstrates strong performance comparable to transformers up to the 2.8B scale, as well as promising scaling laws.

MoE models have been demonstrated to achieve significantly higher performance in both training and inference per FLOP than the equivalent dense models (3; 4). Moreover, scaling laws for MoE models have been put forward (10) which show that MoE performance improves smoothly with compute, data, and the number of experts being routed to. Recently, (5) released a powerful open source mixture of experts model which performs competitively with Llama 2 70B (11) and close to

[2]In this paper, we denote an MoE model with X forward-pass parameters and Y total parameters as X/Y.

GPT-3.5 in evaluations while requiring only the forward pass FLOP cost of the original Mistral 7B model (12), thus demonstrating and solidifying the promise of MoE models at scale.

Concurrently with this work, (13) demonstrate the performance of extremely small mamba-MoE models in the hundred-million scale of total parameters and the forward pass FLOPs of a 25M model, trained on $<$10B tokens. In contrast, we demonstrate empirically the scaling potential and performance of such models at meaningful scales in terms of both parameters and data, by training multi-billion parameter models on 300B tokens.

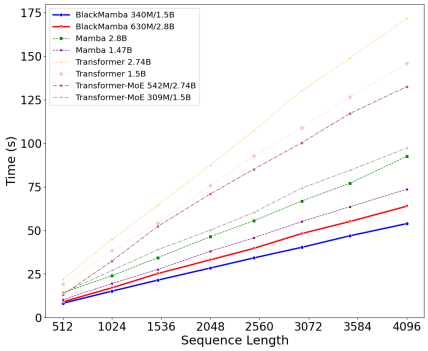
3 DESIGN

A standard transformer model (14) consists of interleaved attention and MLP blocks added in sequence along a residual stream. The equation for a single transformer layer is written in Equation 28. Most MoE architectures simply replace the MLP blocks with a routed expert layer. Our BlackMamba architecture simply replaces both the MLP layer in a transformer with an expert layer, and the attention layer with a mamba SSM layer (see Figure 1). A single block of our architecture can thus be written as,

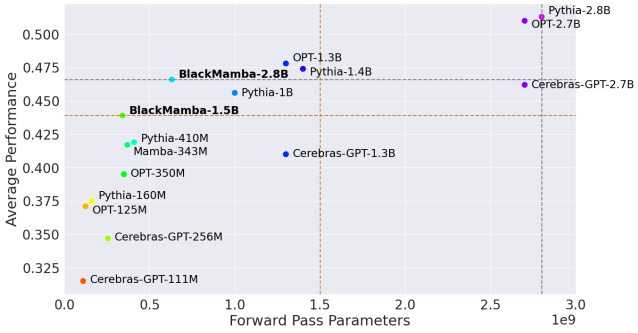
$$x_{l+1} = x_l + \text{MoE}(\text{LN}(x_l + \text{mamba}(\text{LN}(x_l)))) \tag{1}$$

We trained BlackMamba 340M/1.5B and 630M/2.8B models for 300B tokens on our custom dataset. We used the SwiGLU activation function (15) for the expert MLPs. We trained with 8 experts, a number that we found balanced well the trade-off between the inference cost and memory footprint of the model. We tested whether sequential or parallel (16) blocks performed better and found a slight advantage for sequential. Following (11), we trained without biases. For the expert router, we used top-1 routing with a Sinkhorn routing function to load-balance between experts. We utilized a novel custom version of the Sinkhorn algorithm which converges substantially faster than vanilla Sinkhorn (Appendix A.6). We trained using the Megatron-LM (17) distributed training framework. The model was trained in bf16 precision. All further model architectures and training hyperparameters are described in Appendix A.1 and A.2, respectively.

4 RESULTS



(a) Generation latency of BlackMamba compared to dense transformers, dense mamba, and transformer-MoE



(b) Comparison of BlackMamba average evaluation performance across activated forward parameters.

Figure 2: (Left) Generation latency and (Right) evaluation performance as a function of activated forward paramters

To ensure a fair comparison vs Mamba, we trained our own 340M Mamba model with the same dataset and training hyperparameters reported for BlackMamba. This Mamba 340M model used a hidden size of 1152 and 34 mamba layers. Notably, BlackMamba performs significantly better than equivalent pretrained models (both transformer and Mamba) for the same forward pass model size at inference time, as well as training FLOPs. In Figure 2(a), we plot the time taken to autoregressively generate a sequence of a given length starting from an initial one-token prompt as a

function of sequence length. We observe that the established latency benefits of both Mamba and MoE models are combined in BlackMamaba to result in inference times significantly faster than canonical transformer models, MoE transformer models, and pure Mamba models. Moreover, the inference advantage of BlackMamba increases with greater sequence lengths, making BlackMamba extremely competitive at long sequence generation. Moreover, although not reflected in this Figure, it must be recognized that while the transformer inference latency also increases linearly, this is due to KV caching which has additional linearly increasing memory requirements and would eventually OOM on large enough sequences. By contrast, Mamba models (and BlackMamba) can generate sequences of arbitrary length with a constant memory footprint.

In Table 4, we report evaluation scores of BlackMamba against a suite of open-source pretrained language model baselines. We re-evaluated all models on the same version of lm-eval (v0.3.0) that we evaluated our own model on*.

In Appendix A.5, we provide evaluation scores for our model during training from checkpoints taken every 10k steps. We generally found relatively smooth but noisy improvements in the evaluation scores during training. To prevent overfitting to the evaluations, we only looked at the evaluation scores after the models had finished training and did not use them for model selection.

Additionally, in Appendix A.6, we describe a novel initialization for the classical Sinkhorn algorithm used for MoE routing which significantly improves convergence speed of the approach, often requiring only a single iteration for convergence. This provides notable speed improvements for the routed expert layers and results in a similar latency to a router with a regularized balancing loss, providing superior balancing performance while requiring much less complexity of implementation.

Finally, in Appendix A.3, we provide a detailed mathematical description of the internal computations of a Mamba Block and in Appendix A.4, we provide detailed and explicit formulas for computing the parameters and training FLOPs for Mamba and MoE models which we hope aid the community in further developing and exploring novel SSM and MoE architectures.

5 CONCLUSION

In this paper, we have proposed, implemented and trained BlackMamba, a model that combines both recent advances in state-space models and mixture-of-experts into a single unified architecture. We demonstrate that our BlackMamba architecture performs highly competitively to strong pretrained LLM baselines in terms of inference cost and training flops, and moreover that it inherits the reduced training and generation FLOPs of both SSMs and MoEs simultaneously. Moreover, we show that BlackMamba is capable of rapid generation with both linear time and memory cost. We release BlackMamba 340M/1.5 and 630M/2.8 billion parameter models and intermediate checkpoints, as well as inference code, under a permissive Apache 2.0 license with the goal of enabling and fostering further study, experimentation, and understanding of the potential of this novel architecture by the broader community.

*We use the non-normalized HellaSwag evaluation results in this paper, which differs from those in (1)

	Forward Pass Parameters	Total Parameters	Training FLOPs	HellaSwag	PIQA	WinoGrande	Lambada	ARC-e	ARC-c	OpenBookQA	Downstream Average
Cerebras-GPT	111M	111M	2.6e18	0.268*	0.594	0.488	0.194	0.38	0.166	0.118	0.315
OPT	125M	125M	4.1e20	0.313*	0.63	0.503	0.379	0.435	0.189	0.166	0.371
Pythia	160M	160M	4.1e20	0.293*	0.627	0.519	0.389	0.452	0.181	0.16	0.375
Cerebras-GPT	256M	256M	1.3e19	0.286*	0.613	0.511	0.293	0.41	0.17	0.158	0.347
BlackMamba	**342M**	**1.5B**	**6.4e20**	**0.365***	**0.690**	**0.526**	**0.493**	**0.561**	**0.241**	**0.196**	**0.439**
OPT	350M	350M	1.1e21	0.366*	0.644	0.523	0.452	0.44	0.207	0.176	0.395
Mamba	343M	343M	8.0e20	0.335*	0.665	0.516	0.453	0.540	0.212	0.198	0.417
Pythia	410M	410M	1.1e21	0.333*	0.668	0.53	0.505	0.504	0.213	0.178	0.419
BlackMamba	**631M**	**2.8B**	**1.2e21**	**0.397***	**0.712**	**0.521**	**0.542**	**0.603**	**0.245**	**0.242**	**0.466**
Pythia	1B	1B	2.2e21	0.376*	0.705	0.545	0.566	0.559	0.243	0.196	0.456
OPT	1.3B	1.3B	3.2e21	0.4537*	0.717	0.595	0.579	0.57	0.234	0.234	0.478
Cerebras-GPT	1.3B	1.3B	2.8e20	0.384*	0.664	0.521	0.462	0.508	0.224	0.166	0.410
Pythia	1.4B	1.4B	3.2e21	0.398*	0.711	0.565	0.604	0.576	0.256	0.204	0.474
OPT	2.8B	2.8B	6.1e21	0.606*	0.738	0.61	0.637	0.609	0.268	0.25	0.510
Cerebras-GPT	2.8B	2.8B	1.1e21	0.488*	0.701	0.559	0.567	0.571	0.246	0.206	0.462
Pythia	2.8B	2.8B	6.1e21	0.451*	0.737	0.612	0.654	0.629	0.288	0.22	0.513

Table 1: Evaluation performance of BlackMamba compared to similar models

REFERENCES

[1] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[2] B. Peng, E. Alcaide, Q. Anthony, A. Albalak, S. Arcadinho, H. Cao, X. Cheng, M. Chung, M. Grella, K. K. GV *et al.*, "Rwkv: Reinventing rnns for the transformer era," *arXiv preprint arXiv:2305.13048*, 2023.

[3] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *The Journal of Machine Learning Research*, vol. 23, no. 1, pp. 5232–5270, 2022.

[4] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale," in *International Conference on Machine Learning*. PMLR, 2022, pp. 18 332–18 346.

[5] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand *et al.*, "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024.

[6] Y. Sun, L. Dong, S. Huang, S. Ma, Y. Xia, J. Xue, J. Wang, and F. Wei, "Retentive network: A successor to transformer for large language models (2023)," *URL http://arxiv.org/abs/2307.08621 v1*.

[7] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," *arXiv preprint arXiv:2006.16668*, 2020.

[8] W. Fedus, J. Dean, and B. Zoph, "A review of sparse expert models in deep learning," *arXiv preprint arXiv:2209.01667*, 2022.

[9] S. Arora, S. Eyuboglu, A. Timalsina, I. Johnson, M. Poli, J. Zou, A. Rudra, and C. Ré, "Zoology: Measuring and improving recall in efficient language models," *arXiv preprint arXiv:2312.04927*, 2023.

[10] A. Clark, D. De Las Casas, A. Guy, A. Mensch, M. Paganini, J. Hoffmann, B. Damoc, B. Hechtman, T. Cai, S. Borgeaud *et al.*, "Unified scaling laws for routed language models," in *International Conference on Machine Learning*. PMLR, 2022, pp. 4057–4086.

[11] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[12] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[13] M. Pióro, K. Ciebiera, K. Król, J. Ludziejewski, and S. Jaszczur, "Moe-mamba: Efficient selective state space models with mixture of experts," *arXiv preprint arXiv:2401.04081*, 2024.

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

[15] N. Shazeer, "Glu variants improve transformer," *arXiv preprint arXiv:2002.05202*, 2020.

[16] B. Wang and A. Komatsuzaki, "Gpt-j-6b: A 6 billion parameter autoregressive language model," 2021.

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

[18] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," *arXiv preprint arXiv:2111.00396*, 2021.

[19] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou, "A framework for few-shot language model evaluation," 12 2023. [Online]. Available: https://zenodo.org/records/10256836

[20] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" 2019.

[21] Y. Bisk, R. Zellers, R. L. Bras, J. Gao, and Y. Choi, "Piqa: Reasoning about physical commonsense in natural language," 2019.

[22] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "Winogrande: An adversarial winograd schema challenge at scale," 2019.

[23] D. Paperno, G. Kruszewski, A. Lazaridou, Q. N. Pham, R. Bernardi, S. Pezzelle, M. Baroni, G. Boleda, and R. Fernández, "The lambada dataset: Word prediction requiring a broad discourse context," 2016.

[24] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the ai2 reasoning challenge," 2018.

[25] T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal, "Can a suit of armor conduct electricity? a new dataset for open book question answering," 2018.

[26] S. Biderman, H. Schoelkopf, Q. G. Anthony, H. Bradley, K. O'Brien, E. Hallahan, M. A. Khan, S. Purohit, U. S. Prashanth, E. Raff *et al.*, "Pythia: A suite for analyzing large language models across training and scaling," in *International Conference on Machine Learning*. PMLR, 2023, pp. 2397–2430.

[27] R. Sinkhorn and P. Knopp, "Concerning nonnegative matrices and doubly stochastic matrices," *Pacific Journal of Mathematics*, vol. 21, no. 2, pp. 343–348, 1967.

[28] J. He, J. Zhai, T. Antunes, H. Wang, F. Luo, S. Shi, and Q. Li, "Fastermoe: modeling and optimizing training of large-scale dynamic pre-trained models," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2022, pp. 120–134.

[29] Y. Elazar, A. Bhagia, I. Magnusson, A. Ravichander, D. Schwenk, A. Suhr, P. Walsh, D. Groeneveld, L. Soldaini, S. Singh, H. Hajishirzi, N. A. Smith, and J. Dodge, "What's in my big data?" 2023.

[30] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima *et al.*, "The pile: An 800gb dataset of diverse text for language modeling," *arXiv preprint arXiv:2101.00027*, 2020.

[31] D. Soboleva, F. Al-Khateeb, R. Myers, J. Steeves, J. Hestness, and N. Dey, "Slimpajama: A 627b token cleaned and deduplicated version of redpajama," 7 2023. [Online]. Available: https://www.cerebras.net/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama

[32] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim *et al.*, "Starcoder: may the source be with you!" *arXiv preprint arXiv:2305.06161*, 2023.

[33] L. Soldaini and K. Lo, "peS2o (Pretraining Efficiently on S2ORC) Dataset," Allen Institute for AI, Tech. Rep., 2023, oDC-By, https://github.com/allenai/pes2o.

[34] Z. Azerbayev, H. Schoelkopf, K. Paster, M. D. Santos, S. McAleer, A. Q. Jiang, J. Deng, S. Biderman, and S. Welleck, "Llemma: An open language model for mathematics," *arXiv preprint arXiv:2310.10631*, 2023.

[35] J. W. Rae, A. Potapenko, S. M. Jayakumar, and T. P. Lillicrap, “Compressive transformers for long-range sequence modelling,” 2019.

A Appendix

A.1 Model Hyperparameters

Hyperparameter	1.5B	2.8B
Number of Layers	30	36
Hidden Size	1152	1472
Number of Experts	8	8
Sequence Length	2048	2048
State Size	16	16
Convolution Dimension	4	4
FFN Hidden Size	3072	3872
Expansion Factor	2	2

Table 2: Architecture hyperparameters for the 340M/1.5B and 630M/2.8B models

A.2 Training Hyperparameters

Hyperparameter	340M/1.5B	630M/2.8B
Learning Rate	0.0002	0.00015
Batch Size	2064384 tokens	2162688 tokens
Dropout	0.0	0.0
Learning Rate Schedule	cosine	cosine
Min Learning Rate	0.00002	0.00002
Weight Decay	0.0	0.0

Table 3: Training hyperparameters for the 340M/1.5B and 630M/2.8B models

A.3 Mamba Block Internals

In this appendix, we provide a precise and detailed walkthrough of the core computations that comprise a Mamba block. Mamba derives from a line of work on state-space models, which are expressive recurrent models which have recently been shown capable of competing with transformers on large scale sequence modelling. The recurrence of these models enables them to be used efficiently for generation without a KV cache and causes them to scale in FLOPs and memory linearly in the sequence length. The core insight is to utilize recurrence (18) or selective scan (1) to efficiently map the central recurrence to parallel GPU hardware. The base of all such models is the following state-space equations (in continuous time):

$$\frac{dh}{dt} = A\,h + B\,x \tag{2}$$

$$y = C\,h \tag{3}$$

which define a classical linear time-invariant dynamical system. Here h denotes the state of a system at one instant. A denotes a matrix which governs the 'natural dynamics' of h over time. x denotes a 'control' input to the system – i.e. one provided by the controller or experimenter and B denotes a dynamics matrix which controls how x interacts with system. Finally, the states are transformed into 'observations', denoted y, through the observation matrix denoted C.

The Mamba block utilizes this dynamical system across tokens as its core computation implemented as a hardware efficient selective scan. The innovation of Mamba specifically is to make the A,B,and C matrices a linear function of the input x, analogous to the Q,K,V matrices of a self-attention

block. Beyond this, Mamba wraps the SSM component in a linear projection to and from the residual stream and a convolution of the input, as well as an additional gating projection path which gates the output of the SSM based on a projection of the input to the block.

We denote the input to the mamba block x, the recurrent hidden state h, the sequence length as l. We set the hidden recurrent state dimension to some factor of the input dimension.

The mamba block contains matrices A which defines the dynamics for the recurrent state, B which is the projection for the inputs, C which is the projection to the outputs y, the matrix D which is a learnable bias on the output, a discretization timestep dt, and a gating vector z. The Mamba block also performs a linear projection of the input x and z prior to the SSM with weight matrices W_x and W_z and an output projection matrix W_y.

The computation inside a Mamba block runs as follows. First, the x and z projections are computed. This projection occurs for every token in the sequence independently.

$$x = W_x\, x \tag{4}$$
$$z = W_z\, z \tag{5}$$

Secondly, after the projection, the Mamba block performs a 1d convolution ($*$) across the input sequence embeddings. This convolution cannot be merged with the projection W_x because this projection acts at the embedding level, and the convolution is acting at the sequence of tokens level.

$$x_t = W_{filter\_t} * x_t \tag{6}$$

The input-dependent 'weights' B, C, and dt can then be computed, which are analogous to the Query, Key, and Value weights in attention.

$$B = W_B\, x \tag{7}$$
$$C = W_C\, x \tag{8}$$
$$dt = W_D\, x \tag{9}$$

The matrix A is trained with a special initialization given in the matrix below. Note that updates are trained via the parameterization $\ln(A)$, presumably to make A positive and to improve stability, and then computed as $A = \exp(\ln(A))$.

$$A = \begin{bmatrix} 1 & 2 & 3 & \cdots \\ 1 & 2 & 3 & \cdots \\ \vdots & & & \end{bmatrix} \tag{10}$$

The weights are then discretized prior to use in the SSM kernel. Note that the discretization for B does not follow Equation 4 in (1).

$$dt = \text{softplus}(dt + dt_{\text{bias}}) \tag{11}$$
$$dA = \exp(-A\, dt) \tag{12}$$
$$dB = B\, dt \tag{13}$$

A single step of the ssm is then performed to obtain the new recurrent state. Note that $h^+ \to h$ when $dt \to 0$, as expected

$$h^+ = dA\, h + dB\, x \tag{14}$$

From the new recurrent state, the output $C\, h^+$ can be computed. This output is also gated by the learnt gating vector z and passed through a final output projection before being addded back into the

residual stream.

$$y = C\,h^{+} + D\,x \tag{15}$$
$$y = \text{silu}(z)\,y \tag{16}$$
$$y = W_y\,y \tag{17}$$
$$\tag{18}$$

The output of the SSM block is then the hidden state h^{+} and the output y.

A Mamba block can operate in two modes. The first mode is the recurrent method, which directly follows the steps described here. This approach is linear in both memory and computational cost for a single step since it only utilizes the recurrent state to predict the next token. The second way is to run the SSM across the whole sequence at once using the 'selective scan' operation and kernel introduced by (1). For further reference on the implementation of the selective scan refer to (1).

A.4 Computing Parameters and FLOPs for Mamba-MoE

Let us denote the embedding dimension D, the Mamba inner state as I, the recurrent state dimension H, the dt rank dt and the convolution dimension C. We denote the batch size B and the sequence length L.

The number of parameters in a Mamba block can then be computed as,

$$\underbrace{3ID}_{W_x, W_z, W_y} + 2I\Big(\underbrace{H}_{W_A, W_B} + \underbrace{dt}_{W_{dt}} + \underbrace{\frac{C}{2}}_{\text{conv}}\Big) + \underbrace{I}_{D} + \underbrace{2D}_{\text{layernorm}} \tag{19}$$

The number of parameters in a MoE block can be computed as

$$\underbrace{8D^2E}_{\text{experts}} + \underbrace{DE}_{\text{router}} \tag{20}$$

Where E is the number of experts in the layer. For a network of L layers, there are thus $\frac{L}{2}$ Mamba blocks and $\frac{L}{2}$ MoE blocks.

To begin approximating the number of FLOPs involved in a single Mamba block, we make the following observation.

Given two matrices $A \in \mathcal{R}^{K \times M}$ and $B \in \mathcal{R}^{M \times J}$, then the total FLOPs involved in the matrix product AB is approximately $2KMJ$, where the factor of 2 arises from the fact that matrix multiplication requires both a multiply and an add operation. In the following calculations, we assume that the matrix multiplications dominate the total FLOP count of the model and hence ignore the nonlinearities, layernorms, and other computations.

First, let us consider the projection operation involving the weights W_x,W_z, and W_y. All are of shape $I \times D$ and hence the total FLOPs for these are $6IDLB$.

There is also the convolution which can be treated as a single $I \times C$ matrix multiply requiring $2ICLB$ FLOPs.

Now, we turn to the SSM block itself. We first compute the input-dependent B and C matrices requiring a matrix multiply of shape $I \times H$ each thus resulting in $4IH$ FLOPs. The A matrix is not multiplied by the input but goes through an elementwise transform costing IH FLOPs. The dt projection first goes through an elementwise operation of order I FLOPs.

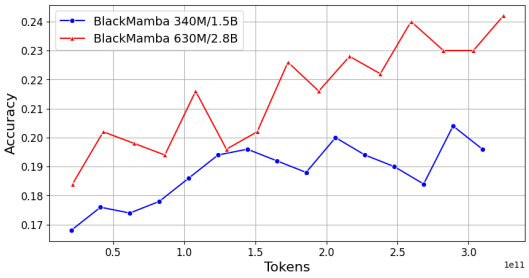
Next, the discretization. The A matrix is multiplied by the dt vector resulting, costing IH FLOPs. The B matrix is multiplied by the input costing $2IH$ FLOPs. The SSM linear state space step itself is just a matrix multiply and add so costs $2IH$ FLOPs, and then the output projection using the C matrix also costs $2IH$ FLOPs. Putting this all together, we obtain the following expression,

$$BLI\Big(\underbrace{11H}_{W_x, W_z, W_y, \text{SSM}} + \underbrace{4dt}_{\text{dt proj, discretization}} + \underbrace{1}_{\text{dt nonlinearity}}\Big) + \underbrace{IH}_{A} \tag{21}$$

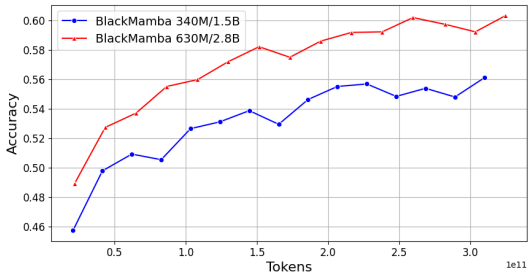
The MoE blocks consist of E standard mlp blocks and a router. The FLOPs for each mlp block is simply $16D^2$ since there are two weight matrices of shape $4D \times D$, and a multiply and add per matrix multiply. The router cost is simply $2DE$. Putting this together, we obtain $DE(16D + 2)$ FLOPs for an MoE block.

A.5 Evaluations During Training

We evaluate BlackMamba on a suite of eight diverse evaluation tasks in the zero-shot setting. We use the EleutherAI evaluation harness (version 0.3.0) (19). Specifically, we evaluate our models on the HellaSwag (20), PIQA (21), WinoGrande (22), Lambada (23), ARC (24) (both the easy and challenge versions), and OpenBookQA (25). The evaluations were run on model checkpoints taken every $10,000$ steps. We observe that most evaluation metrics appear to increase smoothly but noisily throughout training, before appearing to plateau towards their final values. This is broadly in line with previous findings in the Pythia model suite (26), which find relatively smooth improvements across training in many of their evaluation metrics. This provides some evidence that the development of capabilities in language models occurs smoothly and can be tracked during training and perhaps predicted ahead of time. Two evaluation metrics, however, WinoGrande and BoolQ, violate this trend for reasons that we do not currently understand. We note that (26) also observe no consistent trend on Winogrande. Between the BlackMamba $340M/1.5B$ and $630M/2.8B$ models, we observe a clear benefit of scale at the same iteration and token count on most evaluations. In addition, we observe significant noise in some of the evaluation metrics which may suggest that small differences in evaluations between different LLMs may not be significant.

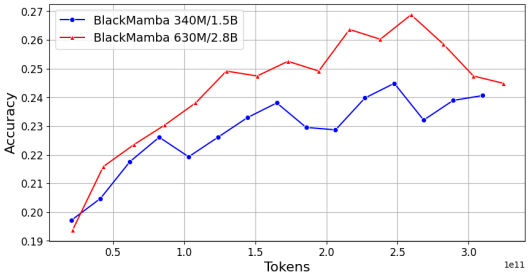


(a) OpenBookQA evaluation accuracy over time

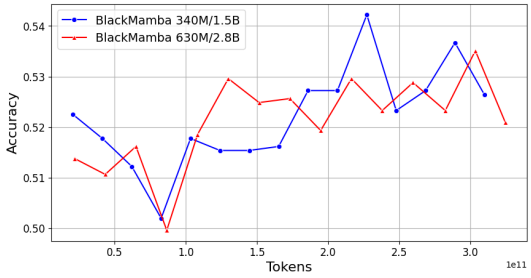


(b) ARC-Easy evaluation accuracy over time

Figure 3: (Left) OpenBookQA evaluation accuracy and (Right) ARC-Easy evaluation accuracy over time

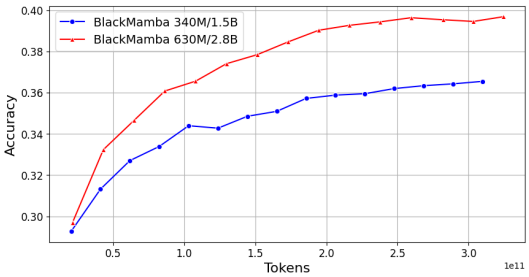


(a) ARC-Challenge evaluation accuracy over time

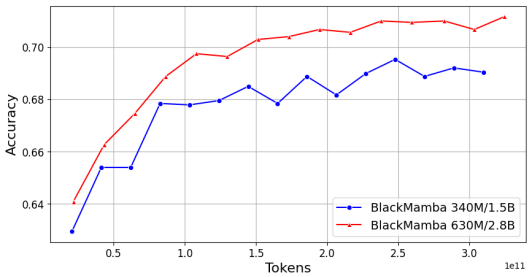


(b) WinoGrande evaluation accuracy over time

Figure 4: (Left) ARC-Challenge evaluation accuracy and (Right) WinoGrande evaluation accuracy over time



(a) HellaSwag evaluation accuracy over time



(b) PIQA evaluation accuracy over time

Figure 5: (Left) HellaSwag evaluation accuracy and (Right) PIQA evaluation accuracy over time

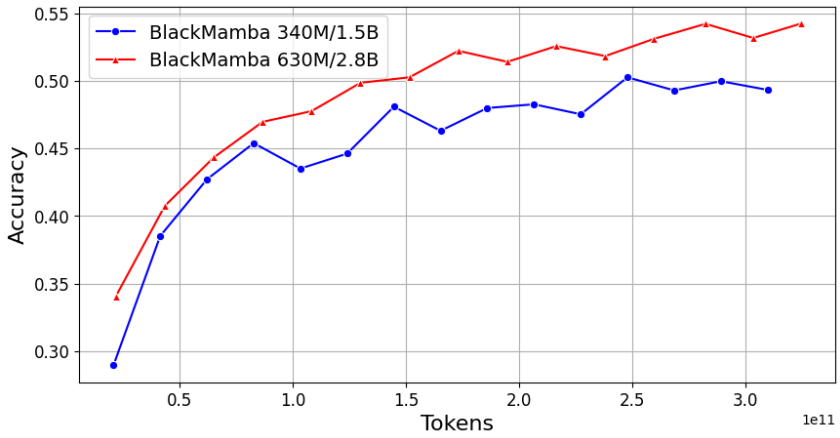


Figure 6: Lambada evaluation accuracy over time

A.6 Sinkhorn MoE Routing Modifications

Recall from the main text eq. (31) that the output token y of an MoE layer is given by

$$y = \sum_{i \in \text{top-}k} c_i E_i(x) \tag{22}$$

where $E_1, E_2, \ldots, E_N$ denote the MLP experts according to the top-k probabilities p_i.

Most commonly, the probabilities $p_i(x)$ are obtained acting by a trainable linear layer on the input $x \in \mathbb{R}^d$ and subsequently applying a non-linearity: $p_i(x) = \sigma(W_i \cdot x)$, with $W_i \in \mathbb{R}^d$. An important issue when training MoE models is that expert utilization should be balanced across tokens in a batch, which is required for compute efficiency. Standard approaches to ensure balanced usage include adding a balancing regularization term to the loss as well imposing hard constraints bounding the number of tokens a given expert can receive (7). We instead use the Sinkhorn activation function for the router which, in the context of top-1 expert selection, has proven to solve the balancing issue without the need for additional regularization or constraints on expert usage (10).

The key property of the Sinkhorn activation function is that, in addition to requiring normalization with respect to the expert index i in $p_i(x)$, one additionally imposes normalization along the samples dimension (which comprises batch size and sequence length). More explicitly, we require that σ satisfies:

$$\sum_{i=1}^{N} \sigma(W_i \cdot x_\alpha) = 1, \qquad \sum_{\alpha=1}^{S} \sigma(W_i \cdot x_\alpha) = S/N \tag{23}$$

where α denotes the sample index, and S is the number of samples (batch size $\times$ sequence length). Now, note that the softmax, which only satisfies the first condition, can be variationally defined by

maximizing:

$$\text{softmax}(L) \equiv \text{argmax}_\pi\{\pi \cdot L + S(\pi)\} \tag{24}$$

where $L_{i\alpha} = W_i \cdot x_\alpha$ are the logits, and $S(\pi) = -\sum_{i\alpha} \pi_{i\alpha} \log \pi_{i\alpha}$ is the Shannon entropy. The Sinkhorn activation can be defined through the same variational formulation except that it further satisfies the second constraint in (23). Denoting the solution to this maximization by

$$\pi_{i\alpha} = e^{L_{i\alpha}} d_i^{(0)} d_\alpha^{(1)} \tag{25}$$

where $d^{(0)} \in \mathbb{R}^N$ and $d^{(1)} \in \mathbb{R}^S$, maximization of the right-hand side of (24) subject to (23) is obtained by solving

$$d_i^{(0)} = \frac{1}{\sum_\alpha e^{L_{i\alpha}} d_\alpha^{(1)}}, \qquad d_\alpha^{(1)} = \frac{S}{N} \frac{1}{\sum_i e^{L_{i\alpha}} d_i^{(0)}} \tag{26}$$

Unfortunately, these equations cannot be solved explicitly and thus, unlike the softmax case, there is no analytic form for the Sinkhorn activation. These equations are solved approximately through an optimization loop, called the Sinkhorn algorithm (27).[3] Our improvement is in the choice of the initial condition for this optimization loop, which consists of taking $d_i^{(0)} = 1$ and $d_\alpha^{(1)} = \frac{S}{N} \sum_i e^{L_{i\alpha}}$. This corresponds to initializing $\pi_{i\alpha}$ to be the softmax normalized along the sample index α, thus immediately guaranteeing balanced usage of experts. We verified empirically that choosing this initial condition leads to much faster convergence of the Sinkhorn loop. Additionally, a temperature rescaling $L_{i\alpha} \to 2L_{i\alpha}$ further improves convergence. Overall this led to shrinking the number of iterations from 10-20 to just 1 across various models sizes, thus shortening the iteration time in our training experiments.

A.7 BlackMamba Routing Profiling

Figures 7(a) and 7(b) illustrate the token counts assigned to each expert in each layer of the BlackMamba 340M/1.5B and the BlackMamba 630M/2.8B models respectively. Most layers display a high degree of expert balance, as expected by our improved Sinkhorn algorithm. Yet, intriguingly, both models show a clear transition towards expert imbalance in the final layers (at layer 20 for the 340M/1.5B model and layer 25 for the 630M/2.8B model). This may reflect increasing specialization in later layers or else reflect numerical instabilities that develop deeper in the network. While the true cause of this imbalance remains unknown, we also note that a similar pattern of imbalance but convergence to a stable expert assignment has also been observed in previous MoE models (28).

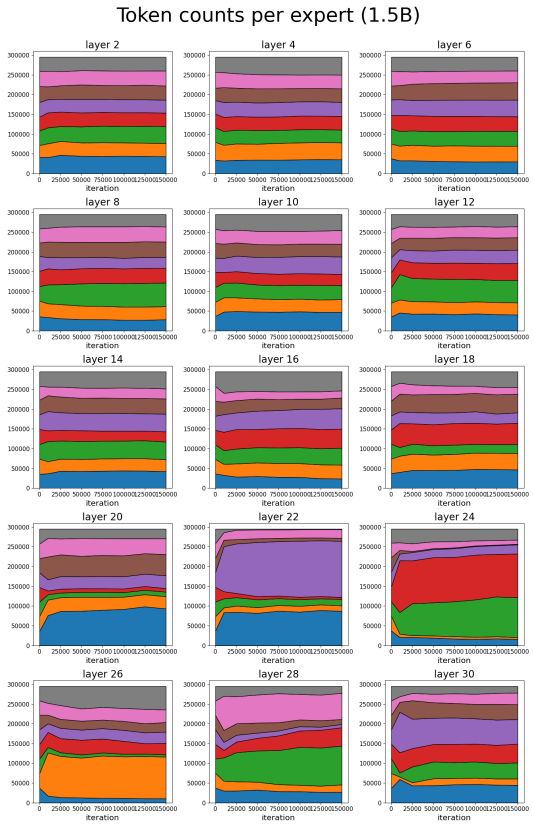
A.7.1 Limitations, Discussion, and Future Work

This work is a preliminary exploration and validation of the core concept of combining together recent advances in SSMs with MoEs to produce a highly competitive and efficient architecture both in terms of inference and generation time and training FLOPs. While initial results are promising, much work needs to be done to improve both the SSM and MoE components as well as investigation of the optimal way to approach their combination. We ultimately believe that by exploring promising emerging architectures architectures and novel ways of merging and combining them, significant advances in performance, efficiency, and speed can be obtained over standard transformer recipes.

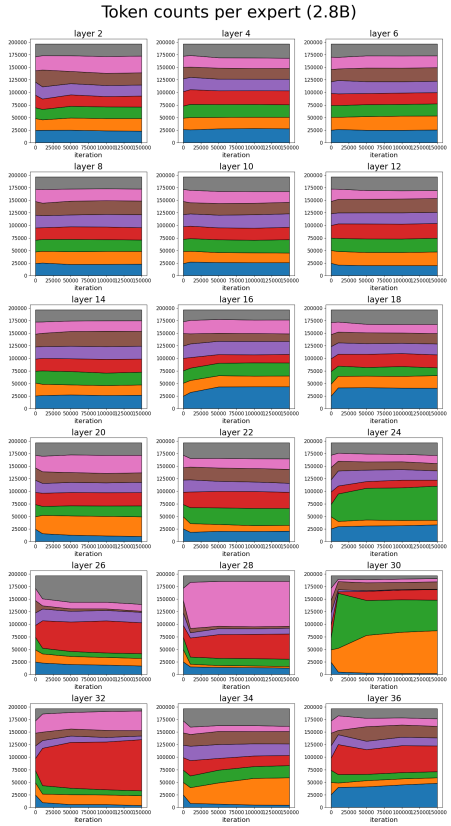
We believe that our work can be extended in many fruitful directions. The evaluations presented in this paper are limited in scope. While we provide general coverage of standard pure language modelling evaluations in the zero-shot setting, the performance of the model in the many-shot in-context-learning setting remains unexplored. Additionally, there are many facets of behaviour of our models which we have not explicitly investigated. We have not tested for factual accuracy, profanity, toxicity, or any other socially undesirable text generation. Similarly, our training dataset blend has not been explicitly scraped for socially undesirable tokens, nor its potential overlap with any evaluation tasks[4]. Although our dataset remains imperfect, we have released all major details as

[3] We need to additionally choose c_i. One natural choice is $c_i = p_i$, but with the Sinkhorn activation we verified that it is more efficient to choose $c_i = f(W_i \cdot x)$ with f a simple activation function such as the sigmoid. We think this is due to the Sinkhorn flattening out more quickly than e.g. sigmoid or softmax due to normalization along both dimensions.

[4] In particular, we are aware of the possibility of evaluation dataset contamination present in the widely used RedPajama dataset (29), and will attempt to explicitly deduplicate this dataset if used in future work.



(a) Token distribution across experts in 340M/1.5B BlackMamba



(b) Token distribution across experts in 630M/2.8B BlackMamba

Figure 7: MoE routing profiles for BlackMamba

to its construction and composition with the goal of aiding community understanding of the effects of dataset on pretraining performance and model behaviours.

In terms of scaling laws, while our models are highly competitive for a given inference cost and FLOP training budget, it is impossible to make conclusive scaling extrapolations both in terms of data and parameter counts with only two models trained on 300 billion tokens. Additionally, many of our training hyperparameters may be suboptimal as we performed only basic hyperparameter tuning of the learning rate. Additionally, while we performed some ablations on the core architecture, it is possible that a superior method of combining state-space models and mixture of experts would provide significant benefits. Additionally, the efficacy and performance of well-established finetuning and RLHF pipelines for instruction following and general alignment, as well as standard techniques for parameter-efficient-finetuning of SSM and MoE models remains almost completely unexplored, as does how such models perform under quantization.

Our work also raises interesting questions as to the modularity of different neural network components that can be placed together into a final model architecture. We show that it is relatively straightforward to combine SSM blocks with MoE blocks from transformers at scale with competitive performance. However, whether Mamba and other SSMs show the same degree of improvement in performance with MoE as transformers remains uncertain, as well as whether combining these architectural pieces has the same effect on the internal representations and behaviours of the model. Additionally, it is unclear the extent to which routing serves the same function in BlackMamba as in more classical transformer MoE models.

A.8 DATASET

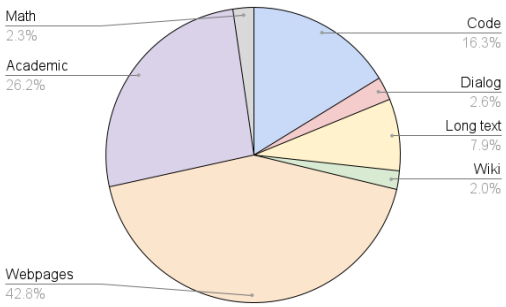


Figure 8: Ratio of data categories in the pretraining dataset of BlackMamba

Dataset	Tokens	Weight
Pile (30)	300B	2
SlimPajama (31)	600B	1.2
Starcoder (32)	250B	0.75
PeS2o (33)	50B	5
Proofpile (34)	40B	2
PG19 (35)	2.2B	5

Table 4: Dataset subsets and their respective weights in our training mixture

To train BlackMamba, we constructed a custom dataset comprised of a mixture of existing open-source datasets. The subsets included: The Pile (30), SlimPajama (31), Starcoder (32), PeS2o (33), and ProofPile (34). The weights for each dataset is provided in Table 4. Tokens were sampled without replacement from each of the subsets according to the probability of sampling from a subset upweighted by these weights. The total dataset comprised 1.8 trillion tokens and thus we trained

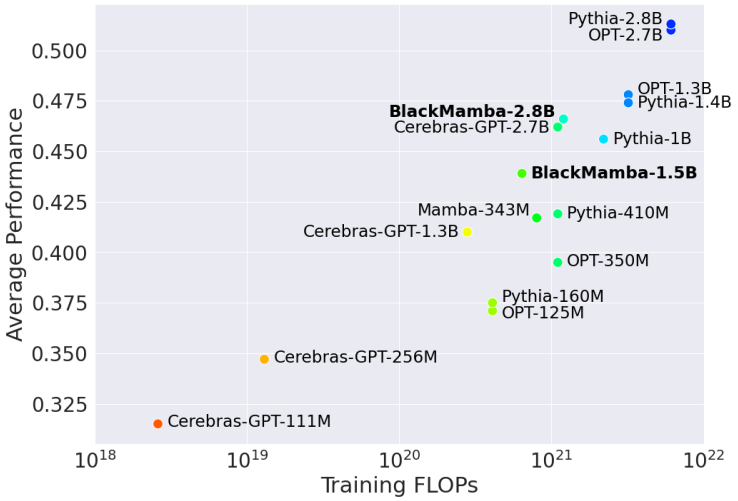


Figure 9: Comparison of BlackMamba average evaluation performance across training FLOPs.

for significantly less than a single epoch. Preliminary experiments[5] show that long-form text and academic work appears to improve natural language modeling when included in the pretraining phase, so we weigh it heavily in the training recipe. Further, we find that including significant portions of code and math during the pretraining phase meaningfully improves the model's reasoning ability. We note that this dataset is comparatively heavy on unfiltered web data and contains many duplicates due to the upweighting of smaller subsets, which may limit the quality of the model and leaves significant room for improvement, as well as potentially causing undue memorization of specific common fragments.

A.9 Background

A.9.1 Transformers

The transformer architecture (14) has demonstrated exceptionally strong and consistent performance at language modelling, as well as almost all other sequence processing tasks, remaining state-of-the-art and essentially unchanged since its introduction. The core operation of the transformer is self-attention, which performs a quadratic all-to-all comparison of the dot-product similarities between the embeddings of different tokens in a sequence before normalizing it and performing a linear map to an output vector. Mathematically, self-attention can be written as,

$$z = W_V x \sigma(\frac{1}{\sqrt{d}} x W_Q W_K^T x \circ M) \tag{27}$$

Where σ denotes the softmax function, M denotes a binary mask which enforces specific constraints, such as causal masking, on the computation, the superscript T denotes transposition, and $\circ$ denotes element-wise multiplication. The quadratic cost in sequence length is caused by the $xW_QW_K^Tx$ term which computes a $L \times L$ matrix of similarity scores between the embeddings of different tokens where L is the sequence length.

The transformer model consists of a stack of self-attention blocks interleaved with multi-layer-perceptron (MLP) blocks which consist of a two-layer MLP with a given activation function. A layer of a transformer model can thus be written as,

$$x_{l+1} = x_l + \text{MLP}(\text{LN}(x_l + \text{attention}(\text{LN}(x_l)))) \tag{28}$$

Where LN represents the layernorm operation which is used to normalize the inputs to the attention and MLP blocks.

[5]We believe that such experiments are not yet rigorous enough for publication, and will be included in future work.

A.9.2 MAMBA

State-space models (SSMs) are a class of sequence models that possess linear complexity with respect to the sequence length. SSMs are more closely related to RNN and CNN architectures than the attention mechanism, and draw inspiration from a continuous dynamical system (depicted in Equation 29) mapping a 1-dimensional function or sequence $x(t) \in \mathbb{R} \mapsto y(t) \in \mathbb{R}$ through an implicit latent state $h(t) \in \mathbb{R}^N$:

$$h'(t) = Ah(t) + Bx(t), \quad y(t) = Ch(t) \tag{29}$$

Where the 'time' t now represents the sequence position of a token. A linear dynamical system like this can be efficiently computed in parallel via a convolution or associative scan, while the recurrent form presented above can be utilized for rapid generation at inference time. The fundamental innovation of the Mamba architecture is to make the A, B, and C matrices of the SSM linearly input-dependent. That is, the new dynamics can be written as,

$$h'(t) = A(x(t))h(t) + B(x(t))x(t), \quad y(t) = C(x(t))h(t) \tag{30}$$

Intuitively, this enables the updates to the SSM's recurrent state to selectively depend upon the tokens being processed, with the SSM being able to decide to store or remove specific information from its recurrent state dynamically. This renders the A,B,C matrices loosely analogous to the Q,K,V matrices in attention and significantly increases the expressivity of the SSM block and could potentially enable context to persist much longer in the hidden state than otherwise, since it must exponentially decay in a linear dynamical system with fixed weights. Empirically, (18) found that this closed much of the gap with transformers.

In practical terms, the recurrent nature of SSMs has long prevented their adoption on the reigning highly-parallel AI hardware like GPUs. However, recent implementations of recurrent and state-space models such as Mamba (1) and RWKV (2) have mapped these operations efficiently to GPU hardware via parallel scan kernels, thus enabling training of such novel architectures with efficiencies approaching that of well-optimized transformer models.

For more details on Mamba, please see Appendix A.3 which describes in details the internal computations of a Mamba block as well as (1) and its associated codebase.

A.9.3 MIXTURE OF EXPERTS

Mixture of Expert (MoE) models allow for the inference cost and number of parameters of a model to be decoupled by not activating all parameters on the forward pass and instead routing tokens to specific MLP *experts*. Each expert theoretically specializes in a certain kind of input, and the router (a small neural network) learns which expert to route each token to. Theoretically, this enables the model to maintain almost all the expressivity of the parameter-equivalent dense model at significantly fewer FLOPs.

In standard implementations (3), which we follow in this paper, the router is a linear layer mapping from tokens to expert indices, and each expert is simply a standard transformer MLP. The expert that the token is routed to is chosen as the top-k of the expert probabilities, where k is a hyperparameter of the architecture. Given an input token to the MoE layer x, this is mapped through the router to a probability distribution $p_i(x)$, where i labels the experts. Upon selecting the top-k probabilities, the output of the MoE layer y can be expressed, schematically, as,

$$y = \sum_{i \in \text{top-}k} c_i E_i(x) \tag{31}$$

where $E_1, E_2, \ldots$ denote the MLP experts,

$$E_i(x) = W_{\text{out}} f(W_{\text{in}}(\text{LN}(x)) \tag{32}$$

where f is the activation function of the MLP, and c_i are coefficients that are often identified with p_i, the probability output by the router of choosing a specific expert. The optimal method for training the router is still uncertain since the "correct" expert assignment problem is non-differentiable, and MoE models often struggle with training stability and load-balancing between different experts for hardware efficiency. Nevertheless, MoE models have demonstrated the ability to achieve superior performance for a given compute budget over dense transformer models. Lastly, due to complexity

of reporting MoE models, where different papers have reported either the forward pass size of the MoE, the total parameters, or both, we here present a consistent convention of denoting MoE models as: $(\text{forward parameters})/(\text{total parameters})$. For more details on the MoE architecture and its typical implementation, see (8).