Longe Range Tasks with Mamba

Martin Fluder
Columbia University

Abstract

Since their invention, the domain of large language models (LLMs) has been predominantly governed by Transformers. The Transformer architecture, fundamentally reliant on matrix multiplication, excels in parallelization, making it particularly suited for modern GPUs. This allows for the efficient scaling of deep learning models. The Multi-Head Attention mechanism of vanilla Transformers is extremely powerful in recognizing short-range dependencies and context. However, the attention mechanism has scalability issues as the computational cost grows quadratically, $\mathcal{O}(n^2)$, with their context length n. With increasing demand for larger context size, there is a need for models that can efficiently capture long-range dependences without the need for excessive hardware. Mamba is a novel architecture that leverages a state space approach to achieve linear scaling with context length – akin to recurrent neural networks (RNNs) – while introducing an innovative and parallelizable "selective scan" method. Mamba modifies traditional Structured State-Space Models (SSMs) by incorporating input-dependent dynamics and showing a promising direction for efficiently handling long-range dependencies. In this article, our goal is to delve into and analyze the foundational elements of the Mamba model. We aim to explore its efficacy on a benchmarks designed to test long-range dependencies, called the Long-Range Arena (LRA). We especially try to compare how Mamba performs against Transformer-based architectures.[1]

[1] Our code can be found on GitHub.

Contents

1 Introduction

The Transformer architecture, which leverages Multi-Head Attention mechanisms as introduced in the seminal works [3, 34], has demonstrated remarkable success across a vast spectrum of applications, ranging from language processing tasks, such as machine translation and text generation, to sequence analysis, computer vision and even protein folding. One key reason for the ubiquitous adaption of Transformers is their ability to scale efficiently with increased model size and depth, the volume of data, and the computational power available for training [16]. This scalability led to the era of *massive* models, characterized by unprecedented levels of size [5] and depths [35], which consume immense amounts of data. The principle of more extensive training on ever larger models has become a cornerstone of this era, and where previously intricacies in architectures led to innovation, scalability and engineering data pipelines has taken over.[2] Additionally, the high degree of parallelization and simplicity of operations (mostly matrix multiplication) inherent in Transformers makes them exceptionally compatible with modern GPU technology, further securing their position as state-of-the-art in many deep learning applications.

[2] Many alternative architectures face limitations in performance improvements despite scaling in size and data. An instance of this was discussed in class, where CNNs often achieve better results for smaller datasets and model sizes on vision tasks but get overtaken by Transformers in the regime of large models and volume of data.

The remarkable efficacy of Transformer models in handling NLP tasks can partly be attributed to their capabilities in detecting short-range and contextual relationships. This stems from the model's fundamental approach of evaluating all token-to-token relationships within a given context window. However, a fundamental limitation of this approach is its inherent inability to access information beyond a predetermined, finite window. Expanding the architecture to incorporate a larger context size is a significant challenge because the self-attention mechanism employed by Transformers exhibits a time and memory complexity that scales quadratically with the length of the sequence and, therefore, performs comparatively poorly on long-range tasks [31]. Numerous strategies have been proposed to extend the context-length capability of Transformers (see Section 2 for some further examples).

For instance, in FlashAttention [7] and FlashAttention-2 [6], the authors focus on optimizing the hardware implementation of a Transformer's attention-layer.[3] They are able to significantly reduce memory constraints and achieve notable runtime speedups. Such advancements are crucial to improve the scalability and practical applicability of Transformer models and enable them to process longer sequences more efficiently with reduced computational cost. In particular, the implementation of FlashAttention significantly improved the Transformer's scores on the Long-Range Arena (LRA) benchmark [7].

With ever-increasing demand for larger context windows and some people claiming that longer context windows will be the main path to AGI systems, improving the context-length computational complexity of Transformers becomes imperative. In this article, we investigate a novel architecture, named Mamba, which claims to reduce the context-length computational complexity to $\mathcal{O}(n)$, while retaining – or even surpassing – the powerful predictive and scaling properties of Transformers [11]. Mamba can be viewed as a Structured State-Space model (SSM) [22] and is an iteration of recent progress of applying the ideas of SSMs to deep learning [13, 29]. SSMs are (partially) observed Markov models, where the hidden state evolves according to a Markov process, which is (potentially) conditioned on some inputs (see Appendix B for a brief overview). Notice that the approach of SSMs is reminiscent of recurrent neural networks (RNNs).

The Mamba architecture is claimed to allow for long context windows while preserving similar predictive power and adaptability to Transformers. In the original paper [11], the authors detail experiments on various tasks such as language and DNA modeling, as well as speech generation, showing improvements over previous architectures at comparable model sizes. In follow-up work, the Mamba architecture has been adapted to vision tasks [36], where the authors provide experiments showing considerable improvements in computation and memory efficiency coupled with improved accuracy across various computer vision tasks compared with DeiT [32].

These results draw a convincing picture of the validity of the initial claims of Mamba's power. Nonetheless, in this article, we investigate the claimed powerful long-range capabilities of the Mamba architecture by studying its performance on the Long-Range Arena (LRA) benchmark [31]. We shall compare Mamba's performance on several vision, language and math-based tasks and compare its benchmark results to similar-sized Transformer models, which include rotary positional embeddings (RoPE) and FlashAttention. The Mamba model is an adaptation of several variants of SSMs, which historically performed exceptionally well on LRA [13]; we expect Mamba's performance on LRA to be in the same region as those models.[4]

The remaining parts of this article are structured as follows. We start with a brief overview of the state-of-the-art long-range models, including SSMs, and why they allow for powerful long-range interactions. We then review the Mamba architecture and its key features compared to previous SSMs. We then briefly discuss the relevant tasks in the LRA dataset and present our experiments and results. Finally, we finish with some concluding remarks. In three appendices, we provide more details about our experiments, discuss some more background information on SSMs, and describe the "selective scan," a clever parallelization technique employed by Mamba.

[3] FlashAttention uses a smart way to utilize GPU design to effectively reduce the context-length space complexity down to linear. The computational complexity remains $\mathcal{O}(n^2)$ however.

[4] We will find that Mamba out-performs Transformers on LRA tasks, but falls short of the results of the best "traditional" SSMs.

2 Related Work: Large Context Models

Transformers. Transformer models dominate all types of deep learning models nowadays. Especially in the realm of large language models (LLMs), *efficiently* dealing with longer context windows has become the next frontier. Several interesting approaches attempt to approximate a Transformer block by distributing the work-load across different hosts; see for example [18]. Other recent methods focus on efficiently dealing with positional encodings and slowly extending context lengths during training, while retaining its short-range capabilities [8]. Some of these approaches have shown impressive results, but do not fundamentally address the bottleneck due to asymptotic $\mathcal{O}(n^2)$ dependency on context-length of Transformers.

Alternative Approaches. As a way to address the sequence length scaling inefficiencies of Transformers, several novel modifications of RNN resurfaced. For example, in [24], the authors consider a "parallelizable" version of RNNs. RNNs exhibit linear scaling with respect to context length, but are limited in their efficient training on modern GPUs. The paper adresses this shortcoming, by utilizing a linear attention mechanism.

Structured State-Space Models. More prominently, a recent class of models – so-called "Structured State-Space Models" (SSMs) – have shown to be increasingly effective at dealing with long-range tasks [13].[5] These models are based on the intuition of state space models from other disciplines, and deal with a discretized version of such variants. Subsequent follow-ups to [13] mostly focused on improving training and inference speeds by clever initializations of various matrices; for example, in the paper [9], the authors focus on High-order Polynomial Projection Operators (HiPPO), which are a special class of matrices derived from Legendre polynomials. Replacing the matrix A (see *e.g.* equation (7)) in S4 with a HiPPO matrix significantly improves its learning capabilities [9]. Further simplifications were considered in S4D [15, 12], where they initialize the matrix A to be a particular diagonal approximation of the S4 HiPPO matrix. More recently, in the Simplified State Space Layers for Sequence Modeling (S5) [29], the authors consider a variant of S4 which includes a parallel scan algorithm (see Appendix C) in order to compute the sequence outputs by operating on the initial elements (tuples of state and input) of the sequence. Finally, in the Mamba paper – also known as Selective SSM or S6 – the authors combine several of these ideas and introduce an input dependent model (see Section 3). While previous SSMs were shown to be less expressive than Transformer models in terms of what "functions" they can approximate [37], it was recently proven [1] that the input depencence of Mamba actually make it *more* expressive than Transformer counterparts, *i.e.* in Theorem 1, they prove that a Selective SSM layer (see Figure 1) is able to express all functions a Transformer head can express, while the reverse is not true.

Long-Range Arena. The Long-Range Arena (LRA) benchmark [31] is commonly used to evaluate various types of SSMs, a benchmark where Transformers typically underperform. The LRA tasks (see Section 4) provide a systematic and unified framework for assessing a model's ability to capture long-range dependencies in long input sequences. The LRA includes six tasks with input sequences ranging from 1K to 16K tokens, covering different modalities, including (nested) mathematical expressions, text, and both real and synthetic images.

3 Mamba Architecture

In this section, we briefly review the Mamba architecture, especially emphasizing novel aspects compared to previous SSMs. We refer to Appendix B and references there for an introduction to SSMs.

Mamba is a modification to general SSMs. In particular, the (discretized) state space model equations in (11) get modified such that the (discretized) matrices $\bar{A}$, $\bar{B}$ and C become input-dependent – *i.e.*

[5] We review the Structured State-Space S4 model in Appendix B.

$\bar{A} \to \bar{A}(x_t), \bar{B} \to \bar{B}(x_t)$ and $C \to C(x_t)$ – and the SSM evolution equations become[6]

$$h_k = \bar{A}_t h_{k-1} + \bar{B}_t x_k \,, \tag{1}$$

$$y_k = C_t h_k \,. \tag{2}$$

Here, $\bar{A}_t \in \mathbb{C}^{N \times N}, \bar{B}_t \in \mathbb{C}^{N \times 1}, C_t \in \mathbb{C}^{1 \times N}$ are discretized, complex matrices governing the evolution of the hidden state vector, $h_k \in \mathbb{C}^N$. More generally, we add another dimension for cases in which the input and output vectors are real or complex-valued vectors, *i.e.* $x_k, y_k \in \mathbb{C}^D$. For ease of notation, we restrict to the case $D = 1$ in the current exposition. In principle, one can choose the way these matrices depend on the input. Mamba uses the following simple linear embedding,

$$A_t = A \,, \quad B_t = \mathrm{lin}_N(x_t) \,, \quad C_t = \mathrm{lin}_N(x_t) \,. \tag{3}$$

The (learned) input-dependent discretization, $\Delta^{(t)}$, that maps $A_t, B_t \to \bar{A}_t, \bar{B}_t$, however, adds a non-linearity,

$$\Delta^{(t)} = \mathrm{softplus}\left(\mathrm{lin}_D\left(\mathrm{lin}_R(x_t)\right)\right) \,. \tag{4}$$

This type of down- and up-projection to R, D dimensions, respectively, together with the softplus function, were chosen due to the resulting similarity with RNNs (see Theorem 1 of [11]). Mamba then applies an approximate version of the zero-order hold (ZOH) discretization scheme,

$$\bar{A}_t = \exp(\Delta^{(t)} A) \,, \tag{5}$$

$$\bar{B}_t = (\Delta A)^{-1} \left(\exp(\Delta^{(t)} A) - I\right) \Delta B \approx \Delta B \,. \tag{6}$$

On the left-hand side of Figure 1, we show a schematic depiction of the selective SSM part of the Mamba block. The Mamba authors make several simplifications in their implementation. They pick the matrices A, B, C purely real and choose a diagonal initialization akin to S4D [12]. Furthermore, as mentioned, they use an approximate version of the ZOH discretization scheme.

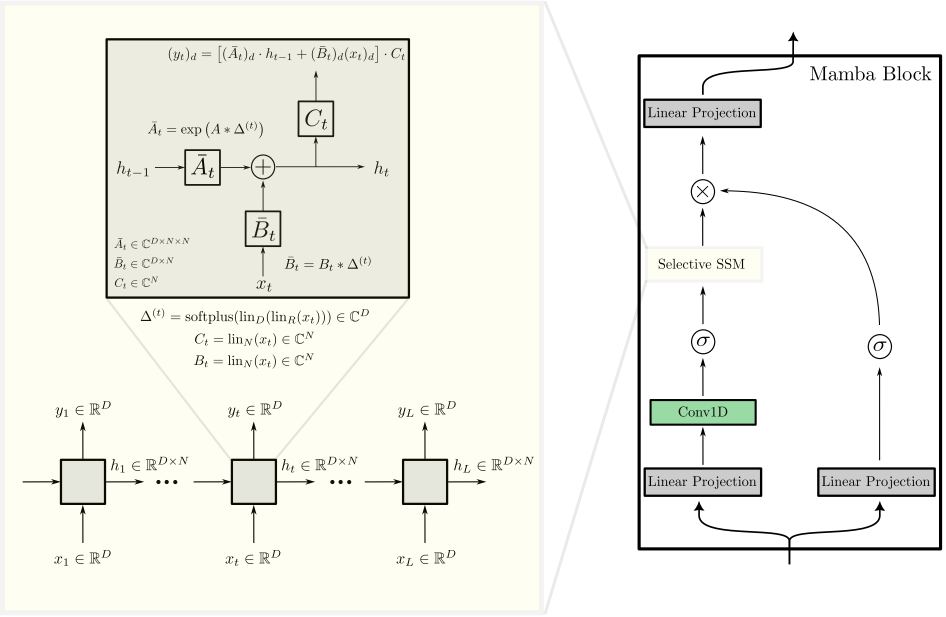
$(y_t)_d = \left[(\bar{A}_t)_d \cdot h_{t-1} + (\bar{B}_t)_d (x_t)_d\right] \cdot C_t$
$\bar{A}_t = \exp\left(A * \Delta^{(t)}\right)$
C_t
h_{t-1}
$\bar{A}_t$
h_t
$\bar{B}_t$
$\bar{A}_t \in \mathbb{C}^{D \times N \times N}$
$\bar{B}_t \in \mathbb{C}^{D \times N}$
$C_t \in \mathbb{C}^N$
x_t
$\bar{B}_t = B_t * \Delta^{(t)}$
$\Delta^{(t)} = \mathrm{softplus}(\mathrm{lin}_D(\mathrm{lin}_R(x_t))) \in \mathbb{C}^D$
$C_t = \mathrm{lin}_N(x_t) \in \mathbb{C}^N$
$B_t = \mathrm{lin}_N(x_t) \in \mathbb{C}^N$
$y_1 \in \mathbb{R}^D$
$y_t \in \mathbb{R}^D$
$y_L \in \mathbb{R}^D$
$h_1 \in \mathbb{R}^{D \times N}$
$h_t \in \mathbb{R}^{D \times N}$
$h_L \in \mathbb{R}^{D \times N}$
$x_1 \in \mathbb{R}^D$
$x_t \in \mathbb{R}^D$
$x_L \in \mathbb{R}^D$
Mamba Block
Linear Projection
Selective SSM
σ
σ
Conv1D
Linear Projection
Linear Projection

Figure 1: Visualization of the Selective SSM (left) and the Mamba block (right) architectures.

Compared to previous SSM architectures, the selection mechanism – *i.e.* the maps $\bar{A} \to \bar{A}_t, \bar{B} \to \bar{B}_t, \cdots$ – is the most crucial addition in Mamba, which is why it is sometimes called Selective SSM.

[6] For SSMs, these matrices are generally considered to be complex. Basic Mamba, however, restricts to real initializations, as they claim that the corresponding real models performs comparable to the complex ones on tasks of most modalities.

This input-dependence, however, naively prohibits computing the output based on a convolutional operation, which is a crucial ingredient allowing for fast inference and training in all previous SSMs, and, in addition, comes with a host of other issues, such as high memory usage. The authors of Mamba cleverly address these problems using three novel ingredients:

(i) Selective Scan.

(ii) Hardware-aware implementation.

(iii) Simplified Mamba Block.

We refer to Appendix C for more details on the selective scan algorithm. The upshot is that it allows for a parallelizable implementation of the selective SSM. Furthermore, Mamba contains a hardware-aware implementation of the selective scan, which heavily reduces the memory consumption.[7] The underlying ideas of this hardware-aware implementation is similar to FlashAttention [7]. Some basic comparisons between a purely (non-hardware optimized) PyTorch version and a (hardware optimized) CUDA version of Mamba show significant differences in speed and memory usage. In our empirical comparisons, we saw an inference and training speed increase by a factor of 5 and decreased memory usage by a factor of over 8.

The structure of the selective SSM is very similar to that of RNNs. However, importantly, the hidden state undergoes a *linear* transformation – matrix multiplication with $\bar{A}_t$ – rather than the usual non-linear $\sigma(Ah_{t-1})$ of RNNs. This linearity precisely allows for parallel and efficient implementation using the selective scan (see Appendix C).

Finally, Mamba introduces a simplified and unified Mamba block (see the right-hand side of Figure 1), which can be stacked akin to the usual Transformer blocks. We notice here that the input gets expanded/copied by (usually) a factor of $E = 2$. The right-hand side that undergoes a non-linearity corresponds to a skip-connection, which coincides with the usual D-term in SSMs (see for example equation 8). The convolutional part is a remnant from H3 and Hyena [10, 25], and its goal is to "smooth out" the input sequence. In Mamba, it is implemented in a hardware-aware fashion, but is not crucial for run-time and memory considerations.

The Mamba architecture combines various ideas and methods from SSMs, RNNs, and Transformers, making it a truly interesting and deep architecture. It would be interesting to further test some of the effects of the simplifications and assumptions made along the way.

4 Long Range Arena Tasks

The primary goal of this article is to perform various stress tests of Mamba for long-range tasks. We focus on the Long-Range Arena (LRA) benchmark [31], which contains six (synthetic) tasks, including a long version of ListOps, byte-level text classification, and document retrieval. The latter two tasks artificially increase the input lengths via a byte-level encoding. Finally, it contains three image tasks (although two are essentially the same with different resolutions): CIFAR-10, Pathfinder, and PathX.

4.1 ListOps

The LRA ListOps task is a longer version of ListOps from [23]. ListOps contains examples of summary operations applied on a nested list of single-digit integers written in prefix notation. The summary operations consist of max, min, med and $\mathrm{sum\_mod}$, *i.e.* taking the maximum, minimum, median, and the sum modulo 10 of the consecutive integers. The model then needs to predict the outcome of these nested operations modulo 10, *i.e.* it is a 10-way classification task. For example, the correct value of the following sequence

$$[\ \mathrm{med}\ 6\ [\ \mathrm{med}\ 3\ 2\ 2\ ]\ 8\ 5\ [\ \mathrm{med}\ 8\ 6\ 2\ ]\ ]$$

would be 6. The LRA ListOps task contains much longer sequences up to a maximum length of 2k elements, and the model needs to be able to access all tokens reaching back to the first operation to produce the correct output.

[7] For more details on the hardware-aware implementation we refer to Section 3.3.2 of [11].

4.2 Text Classification (IMDB)

The text classification task of the LRA consists of the usual IMDB sentiment classification task [19] with the twist that the text is byte/character-level encoded to artificially extend the input sequence. It is supposed to assess a model's ability to make contextual connections across a long input series. This task is, therefore, a binary classification with a sequence length of up to 2K tokens.

4.3 Document Retrieval

The document retrieval task involves byte-level document retrieval, with the objective of measuring how effectively a model can embed long contextual inputs and determine whether it can distinguish or match different documents. The data is sourced from the ACL Anthology Network Corpus (AAN) [27], which contains a citation network of scientific papers. The objective of the model is to decide whether two documents are related, making it a binary classification task. The documents are encoded at the byte/character level and truncated to 4k tokens each, resulting in a total input length of 8k tokens.

4.4 Image Task (CIFAR)

The LRA image task mirrors the greyscale CIFAR-10 task, but instead of the input being a single-channel 32×32 matrix, it is flattened into a sequence of 1×1028. Like the original CIFAR-10 task, this is a 10-way classification problem.

4.5 Pathfinder Tasks (Pathfinder and PathX)

Finally, the LRA pathfinder tasks (Pathfinder and PathX) consist of an artificial image task where a model must determine whether two points are connected by a line [17]. See Figure 2 for examples at different resolutions. To increase difficulty, the images are flattened into sequences of length equal to the number of pixels. The image resolutions vary from 32×32 to 256×256. The "basic" Pathfinder task involves 32×32 pixels, while the "extreme" PathX task uses 128×128 pixels. Two additional tasks, Pathfinder-64 with 64×64 pixels and Pathfinder-256 with 256×256 pixels, are sometimes included, particularly for models with strong long-range capabilities. Due to limited computational resources for this project, we could only run experiments for the basic Pathfinder task and the Pathfinder-64 scenario.

Due to the limited scope and available computational power for this project, we shall specifically try to measure the performance of relatively small Mamba and Transformer models (of a maximum of 600k parameters) on these tasks. It is worth mentioning that the "philosophy" of the LRA is somewhat subtle. The original paper [31] suggests minimal parameter-tuning and stresses to train the models from scratch in order to provide a fair comparison between different models' capabilities. Furthermore, they emphasize that the models should be roughly of equal size ($\pm 10\%$ parameters) to be considered for their comparisons. For example, in a recent paper [2], the authors show that model performance can be significantly boosted upon pre-training on tasks such as next-token-prediction. In the present article, we are careful to provide results of comparable models without any pre-training. Furthermore, we shall keep in mind that when comparing to the literature – specifically models we did not run ourselves – we are comparing to larger, more fine-tuned models that have been trained for much longer.

5 Results and Analysis

Let us now turn to our main results. We mainly focused on setting up and running the Long-Range Arena tasks with Mamba models. At the time of this writing, we have not seen any results of Mamba evaluated on the LRA, so we had to set up these experiments from scratch. We start by presenting some empirical results on training and inference speeds and then present our results for the LRA benchmark. Finally, we discuss a few modifications of Mamba and their resulting performance on some LRA tasks.

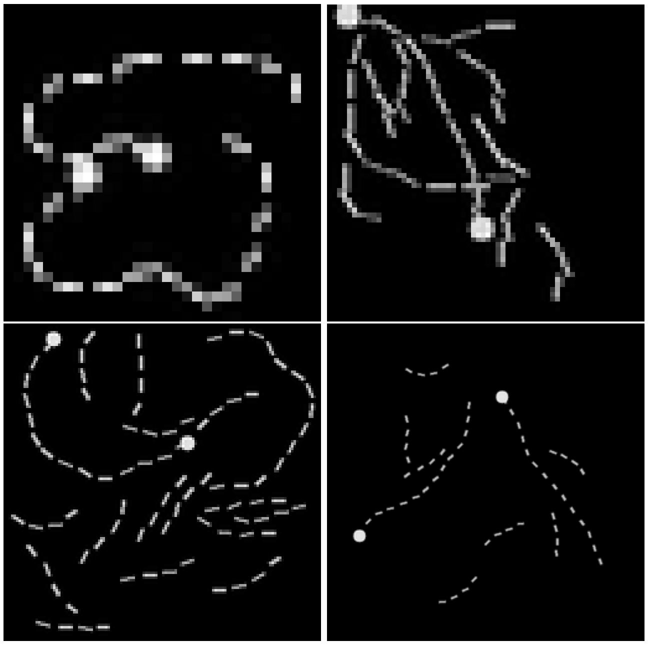

Figure 2: Pathfinder image tasks with increasing resolution; from top to bottom and left to right, 32×32, 64×64, 128×128 and 256×256. Once we unravel the pictures into a sequence, the input lengths of the sequences are 1024, 4096, 16384, and 65536, respectively. The second and third pictures should output a positive answer, while the first and fourth are negative.

5.1 Inference and Training Speed comparisons

We first compare the training and inference speeds of models based on Multi-Head FlashAttention (MHFA) against models based on the Mamba block (see Figure 4 where we compare two examples of such models). We can explicitly observe the quadratic dependency on the input size of the MHFA model, while the Mamba – as expected – scales approximately linearly with increasing input size. These experiments are based on a particular model, as detailed in 2, and ran on Google Colab's NVIDIA A100-SXM4 GPU with 40GB of memory. We ran similar speed tests on various other models and GPUs across (comparable) model dimensions and configurations and found the same qualitative behavior. The present experiments are based on the LRA Pathfinder task, where we consecutively increase the input's resolution.[8] The behavior is in accordance with the results in the Mamba paper [11]. However, we take a more holistic and practical approach, which includes additional overhead such as loading the data, *etc.* and ,therefore, our results will always be (asymptotically lower) bounded by $o(n)$.[9]

5.2 Long-Range Arena Results

In this section, we review our results and the methodologies used to achieve the performance accuracies for LRA tasks as shown in Table 1.

Throughout, we exclusively use RoPE [28] for the Transformer/MHFA based models, as it leads to significant performance boosts (over 10% for the Image task) [20, 2]. Furthermore, for these models, we mainly follow the configurations provided in the literature [31, 20, 2], and depending on their size scale them down to be more in line and comparable to the corresponding Mamba model.

[8] The same behavior continues to the 256×256 resolution, corresponding to input length of 65536 tokens. However, these experiments required smaller batch sizes, differing from those used in our LRA tests.

[9] In the Mamba paper, they compare the isolated block speeds, while here we compare the full model speeds relevant for LRA performance. Their claim is that the block only uses $\mathcal{O}(1)$ inference speed, which we cannot check with our method.

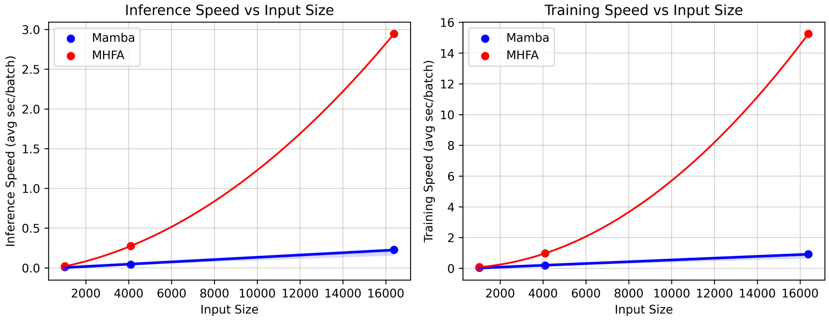

Figure 3: Training and inference speed (average seconds per batch) comparisons of a Multi-Head FlashAttention (MHFA) based model against a Mamba-based one. We fit a linear and quadratic curve to the data, respectively. The underlying model configurations are provided in Table 2.

For the Mamba models, a significant amount of trial and error was involved in determining the optimal parameters and settings for the given tasks. The Image task, in particular, is highly susceptible to overfitting, and large models tend to degrade quickly, even with various regularization methods. In particular, we conducted extensive testing with different configurations, such as varying the number of layers, feature dimensions (D), state dimensions (N), convolutional kernel sizes, optimizers, dropout rates, gradient clipping, learning rates and their schedules, pooling methods, *etc.*. Our search primarily focused on relatively small models with approximately 500K parameters, as this size allowed for a broader exploration within our limited computational budget.

Overall, we are careful to invoke the following guidlines in evaluating our models:

- The best-performing model is picked based on the test accuracy for the model with the maximal validation accuracy.
- We consider models of comparable size, often using larger Transformer models and relatively small Mamba ones.[10]
- We avoid using extensive regularization methods (like dropout, weight decay, and gradient clipping) to minimize overtuning and instead focus on the fundamental capabilities of the models.
- We evaluate a similar number of epochs for the models we are comparing.[11]

We could imagine different philosophies , but these policies were most in line with the ideas of the original LRA paper [31]. See Section 6 for some remarks to this end.

The results of our experiments are summarized in Table 1 and in Figure 4. The configurations of the corresponding (best-performing) models is provided in the Appendix, Table 3. The Mamba models – smaller than their MHFA counterparts except for the Image case – consistently outperform their MHFA counterparts. The difference is relatively small for the Image and Pathfinder-32 tasks. However, specifically for the Pathfinder task, the Mamba model is significantly smaller than its counterpart. For the text (Text and Retrieval) and ListOps tasks, the difference is more significant. A major reason for this is that the Image and Pathfinder-32 tasks start overfitting relatively quickly, so finding better hyperparameter values will result in better-performing models. While this has been done for the Transformer cases in the literature, for the Mamba models, we had to search from scratch and might not have found the best-possible ones provided a fixed number of parameters. Overall, the biggest differences occur for the text tasks, which suggests that – with the above caveat – Mamba might outperform Transformers in NLP tasks. On average, the Mamba models outperform the Transformer ones by approximately 8%, which together with their significant speed improvements provides further evidence of Mamba's potential (see Figure 5).

[10] Sometimes this is due to necessity, as even moderately sized Mamba models were prone to overfit the Image and Pathfinder tasks.

[11] One could imagine that a time-based approach leads to a fairer comparison.

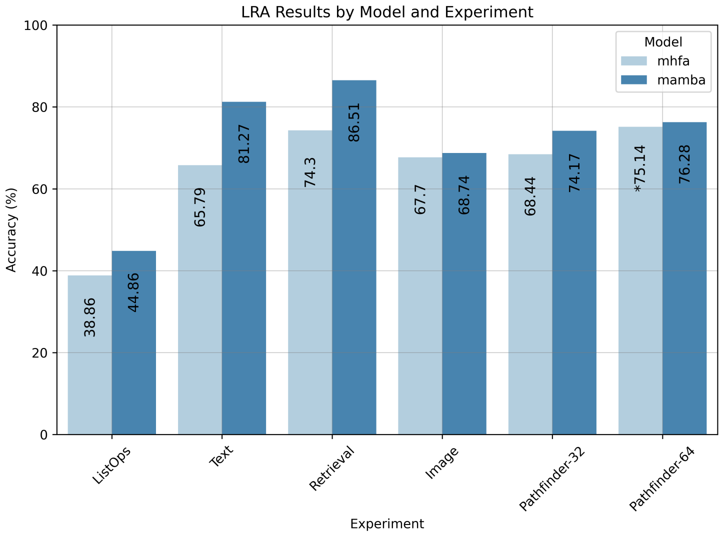

Figure 4: Multi-Head FlashAttention (MHFA) based models versus Mamba-based models: Performance comparison for different LRA tasks.
*We were unable to obtain non-random results for the MHFA model for the Pathfinder-64 task due to limited computational budget, and therefore, we took the results from [20]; the remaining values are based on our experiments.

Model	ListOps	Text	Retrieval	Image	Path	Path-64	PathX	Avg*
(input length)	2000	2048	2×4000	1024	1024	4096	16384	
Mamba	44.86	81.27	86.51	68.74	74.17	76.28	N/A	71.11
MHFA+RoPE	38.86	65.79	74.30	67.70	68.44	N/A	N/A	63.02
Transformer [31]	36.37	64.27	57.46	42.44	71.40	N/A	N/A	54.39
Performer [31]	18.01	65.40	53.82	42.77	77.05	N/A	N/A	51.41
RoPE [26]	40.57	66.53	78.75	62.50	72.68	N/A	N/A	64.59
RoPE [2]**	47.90	79.08	82.31	75.04	76.64	N/A	84.72	72.19
S4 [13]	59.60	86.82	90.90	88.65	94.20	N/A	96.35	84.03
Diagonal [15]	60.6	84.8	87.8	85.7	84.6	N/A	87.8	80.7
S5 [29]	62.15	89.31	91.40	88.00	95.33	N/A	98.58	85.24

Table 1: Summary of the performance accuracies for the LRA Tasks. The first two rows are our results derived from comparable model sizes between MHFA and Mamba-based architectures. The precise configurations of the best-performing models are provided in Table 3 in Appendix A. In order to provide further context, we add LRA results for several Transformer (rows 3-5) and SSM (rows 6-8) architectures from the literature. It is generally hard to determine how tuned such results are and comparison should be done carefully.
*We do not include Pathfinder-64 or PathX in our average computations as these results are incomplete.
**These results were achieved for significantly larger models compared to ours.

In Table 1, we further compare our Mamba LRA results with results from the literature. As mentioned, these results should be taken with a grain of salt. Nonetheless, we remark that Mamba sits relatively consistently in the middle between the best-performing Transformer models and S4 and S5 SSMs. Since Mamba is a modification to these SSMs, we would have expected them to perform more closely to such models (specifically the Diagonal SSM). However, some of these SSMs are significantly larger than the Mamba models considered in this article, and it would be interesting to compare similar-sized models in more depth. We remark here that the jump from diagonal to other SSMs suggest that relaxing the diagonal assumption for Mamba might also help improving its performance. We leave this for future work.

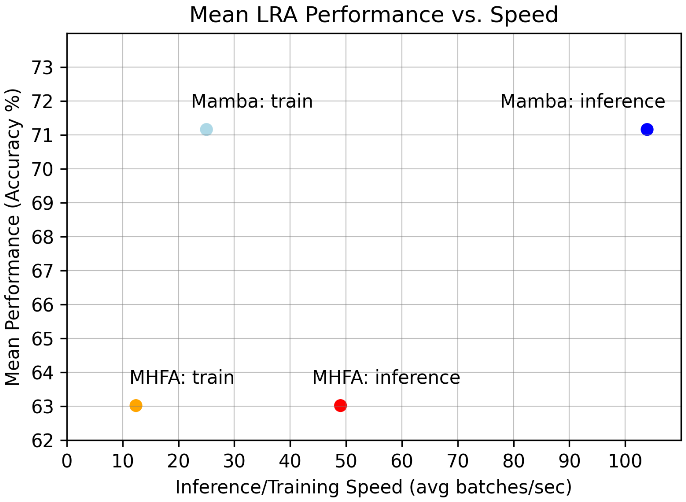

Figure 5: Visualization of training/inference speeds and mean LRA performance for our MHFA and Mamba models.

5.3 Complex Initialization and Other Variations

This section discusses two simple modifications to the basic Mamba block. First, we investigate the effects of complex (instead of real) diagonal initialization of A on the Pathfinder and CIFAR tasks. Secondly, we examine the impact of interleaving Mamba and MHFA blocks.

As mentioned in Section 3, the authors of Mamba make several simplifying assumptions. For instance, they choose A to be a real diagonal matrix. They argue that the complex version generally does worse on language tasks. Here, we investigate whether this applies to the LRA's Pathfinder and CIFAR tasks. We implemented the complexified version and tested it on these two tasks. Changing to complex initialization notably increases the number of trainable parameters of the model, and, therefore, a clean comparison is hard. Nonetheless, our results in Figure 6 seem to suggest that the model with complex A learns faster. However, they seem to asymptote towards the same final performance for these either task. The configurations for this experiment are provided in Table 4. We leave it for future work to investigate this in more detail for the remaining LRA tasks; specifically the language LRA tasks could be viewed as a proxy for their assertion that Mamba with complex A performs worse on NLP tasks.

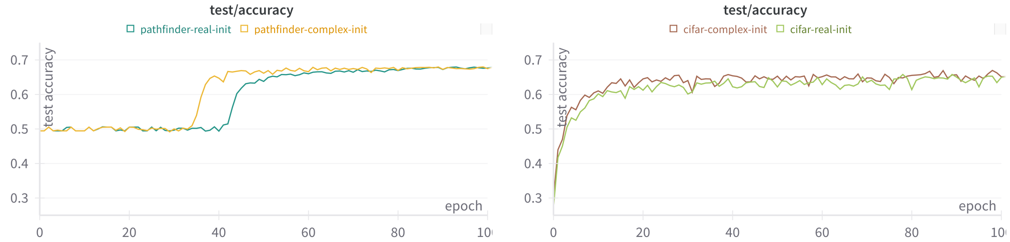

Figure 6: Comparison of test accuracies per epoch for Mamba models with real and complex A.

Additionally, we performed several experiments comparing stacking Mamba with Transformer blocks against pure Mamba versions. While this exercise was interesting, we did not find any setup that significantly improved Mamba's performance on the Pathfinder and CIFAR tasks. It is notable that these models consistently performed similar to pure Mamba rather than MHFA. Furthermore, adding MHFA blocks considerably affected training and inference speeds.

6 Conclusions and Future Work

In this article, we explored Mamba's performance on long-range tasks, which were designed to evaluate a model's ability to efficiently learn long-range dependencies. We focused on comparing the Mamba block with Multi-Head FlashAttention (MHFA), ensuring the comparison was fair and not skewed by tuning a single model for optimal results. Mamba's nearly 10% improvement across the LRA tasks, compared to the commonly used MHFA with RoPE, along with its notable speed enhancements, indicates that Mamba could be a powerful new architecture capable of competing with the ubiquitous Transformer, especially in tasks requiring large context windows like DNA Sequence Modeling [28]. We also examined slight modifications to Mamba's initialization (real versus complex) and interleaving Mamba blocks with Transformer blocks.

We came up with this project idea when learning about FlashAttention and only discovered later that Mamba got rejected from ICLR partly due to a lack of LRA performance results. While Mamba performs well against Transformers, it does worse against other SSMs. Going into this project, we expected Mamba to perform closer to S4. However, the purpose of the LRA benchmark is somewhat questionable. Although it measures a model's ability to capture long-range dependencies, factors like task-specific over-tuning, pre-training, inconsistencies across model sizes, and other adjustments can significantly skew the results.

In [2], the authors found that pre-training Transformer models with RoPE could yield results competitive with SSMs such as S4. They suggest that pre-training offers a fairer comparison of different models' capabilities, arguing that the optimal point in a model's parameter space speaks more to its underlying potential than training from scratch on a single task. While the validity of this claim is debatable, pre-training is a common practice in many real-world applications. It would be interesting to explore whether pre-training Mamba could lead to a performance boost.

For future work, it would be interesting to explore Mamba's performance across larger models with more extensive training, allowing for a more comprehensive comparison of its capabilities with those reported in the literature. Additionally, it would be intriguing to investigate generalizing to off-diagonal matrix structures like HiPPO or DPLR, incorporating fully complex models, experimenting with various discretization schemes, and exploring additional modifications to the Mamba block. It would also be valuable to assess how a bi-directional Mamba version would perform on the LRA tasks, see for example [28], in which the authors introduce BiMamba.

Acknowledgements

I would like to thank Prof. Belhumeur for delivering an incredibly interesting, engaging, and exciting course throughout the semester and for providing the foundation for my work on Mamba. Additionally, I extend my thanks to the TAs for their assistance in locating a functional GPU in GCS.

Contributions

The following are the contributions of this article:

- We implement Mamba for the following LRA benchmark tasks from scratch: ListOps, Text, Retrieval, Image, Pathfinder-32, Pathfinder-64. This required us to experiment with model settings, model compositions that worked best for these tasks, as well as understand the underlying architecture and its relation to SSMs.
- We further run these experiments on the best-performing Transformer models to compare the performance between the two.
- We extract various properties (expecially training, inference speed, memory usage) of these models and make comparisons.

- We experiment with a variety of small modifications to Mamba (complex initialization, layered Mamba-Transformer blocks, *etc*) and assess its performance impact on some LRA tasks.

References

[1] Ameen Ali, Itamar Zimerman, and Lior Wolf. The hidden attention of mamba models. *arXiv preprint arXiv:2403.01590*, 2024.

[2] Ido Amos, Jonathan Berant, and Ankit Gupta. Never train from scratch: Fair comparison of long-sequence models requires data-driven priors. *arXiv preprint arXiv:2310.02980*, 2023.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Guy E Blelloch. Prefix sums and their applications. In *Sythesis of parallel algorithms*, pages 35—60. Morgan Kaufmann Publishers Inc., 1990.

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

[6] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[7] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[8] Yiran Ding, Li Lyna Zhang, Chengruidong Zhang, Yuanyuan Xu, Ning Shang, Jiahang Xu, Fan Yang, and Mao Yang. Longrope: Extending llm context window beyond 2 million tokens. *arXiv preprint arXiv:2402.13753*, 2024.

[9] Daniel Y Fu, Tri Dao, Khaled K Saab, Armin W Thomas, Atri Rudra, and Christopher Ré. Hungry hungry hippos: Towards language modeling with state space models. December 2022.

[10] Daniel Y Fu, Tri Dao, 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*, 2022.

[11] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[12] Albert Gu, Karan Goel, Ankit Gupta, and Christopher Ré. On the parameterization and initialization of diagonal state space models. *Advances in Neural Information Processing Systems*, 35:35971–35983, 2022.

[13] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

[14] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems*, 34:572–585, 2021.

[15] Ankit Gupta, Albert Gu, and Jonathan Berant. Diagonal state spaces are as effective as structured state spaces. March 2022.

[16] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[17] Drew Linsley, Junkyung Kim, Vijay Veerabadran, Charles Windolf, and Thomas Serre. Learning long-range spatial dependencies with horizontal gated recurrent units. *Advances in neural information processing systems*, 31, 2018.

[18] Hao Liu, Matei Zaharia, and Pieter Abbeel. Ring attention with blockwise transformers for near-infinite context. *arXiv preprint arXiv:2310.01889*, 2023.

[19] Andrew Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pages 142–150, 2011.

[20] Yuzhen Mao, Martin Ester, and Ke Li. Iceformer: Accelerated inference with long-sequence transformers on CPUs. In *The Twelfth International Conference on Learning Representations*, 2024.

[21] Eric Martin and Chris Cundy. Parallelizing linear recurrent neural nets over sequence length. *arXiv preprint arXiv:1709.04057*, 2017.

[22] Kevin P. Murphy. *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023.

[23] Nikita Nangia and Samuel R Bowman. Listops: A diagnostic dataset for latent tree learning. *arXiv preprint arXiv:1804.06028*, 2018.

[24] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

[25] Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. In *International Conference on Machine Learning*, pages 28043–28078. PMLR, 2023.

[26] Zhen Qin, Weixuan Sun, Kaiyue Lu, Hui Deng, Dongxu Li, Xiaodong Han, Yuchao Dai, Lingpeng Kong, and Yiran Zhong. Linearized relative positional encoding. *arXiv preprint arXiv:2307.09270*, 2023.

[27] Dragomir R. Radev, Pradeep Muthukrishnan, Vahed Qazvinian, and Amjad Abu-Jbara. The acl anthology network corpus. *Language Resources and Evaluation*, 47(4):919–944, 2013.

[28] Yair Schiff, Chia-Hsiang Kao, Aaron Gokaslan, Tri Dao, Albert Gu, and Volodymyr Kuleshov. Caduceus: Bi-directional equivariant long-range dna sequence modeling. *arXiv preprint arXiv:2403.03234*, 2024.

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

[30] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[31] 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. *arXiv preprint arXiv:2011.04006*, 2020.

[32] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

[33] Arnold Tustin. A method of analysing the behaviour of linear systems in terms of time series. 1947.

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

[35] Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Dongdong Zhang, and Furu Wei. Deepnet: Scaling transformers to 1,000 layers. *arXiv preprint arXiv:2203.00555*, 2022.

[36] Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. Vision mamba: Efficient visual representation learning with bidirectional state space model. *arXiv preprint arXiv:2401.09417*, 2024.

[37] Itamar Zimerman and Lior Wolf. On the long range abilities of transformers. *arXiv preprint arXiv:2311.16620*, 2023.

A Experimental Configurations

In this Appendix, we provide the configurations for the speed tests in Table 2, for the best performing Mamba and MHFA-based models on the LRA benchmark in Table 3, as well as the complex and real Mamba models in Table 4.

We provide various hyperparameters for the models, including the number of *trainable* parameters. In general, we pool the outputs by attaching a decoder to the model, which applies a non-linear two-layer MLP to the pooled output from the MHFA/Mamba model. Depending on the task, we used either an embedding (emb) or a simple linear (lin) layer for the encoder. Adding an FF (feedforward) layer did not significantly improve the performance of most Mamba models, while it did for the Transformer models.

All our MHFA-based models contain rotary positional embeddings (RoPE), which significantly boosts their performance on long-range tasks [30] compared to positional embeddings.[12]

Model	Input Size	Parameters	feature dim	Layers	$h/(N,E,d)$	FF Size	Batch Size
Mamba	1024	614K	128	4	(64, 2, 4)	N/A	32
MHFA	1024	598K	128	6	8	128.0	32
Mamba	4096	614K	128	4	(64, 2, 4)	N/A	32
MHFA	4096	598K	128	6	8	128.0	32
Mamba	16384	614K	128	4	(64, 2, 4)	N/A	32
MHFA	16384	598K	128	6	8	128.0	32

Table 2: Model parameters for the speed comparison in Figure 4. Here, h corresponds to the number of MHFA-heads, N to Mamba's hidden state dimension as in Figure 1, $E = 2$ to the expansion factor, and $d = 4$ to the Conv1D kernel size.

B Review of Structured State Space Models

In this Appendix, we shall provide a brief overview over Structured State Space Models (SSMs), as Mamba is built on its basis. We shall focus on the S4 formulation of SSMs [13], as it provides a foundational starting point for subsequent models.

State space models (SSMs) are inspired by a fundamental physics problem. Namely, in the standard formulation of classical physics, we generally aim to solve the equation force = mass*acceleration. This can succinctly be written as

$$F = m\ddot{x}(t)\,,$$

where $\ddot{x}(t)$ is the second derivative of the position of our point particle. We can add various forces that depend on the velocity, $\dot{x}(t)$, of the particle or the position of the particle on the right-hand side,

[12] A basic comparison shows an improvement of over 10% on the LRA Image (CIFAR) task.

Task	Model	Params	dim	L	$h/(N, E, d)$	FF	Pool	Enc	LR	WD	BS	GC	Epochs
ListOps	Mamba	634K	128	4	(64, 2, 4)	N/A	pool	emb	1e-4	0.05	32	0.0	25
ListOps	MHFA	797K	128	4	8	512	pool	emb	1e-3	0.00	32	0.0	25
Text	Mamba	648K	128	4	(64, 2, 4)	N/A	pool	emb	1e-4	0.10	32	0.0	45
Text	MHFA	827K	128	4	8	512	pool	emb	1e-4	0.10	32	0.0	45
Retrieval	Mamba	692K	128	4	(64, 2, 4)	N/A	pool	emb	1e-4	0.05	32	0.0	10
Retrieval	MHFA	871K	128	4	4	512	pool	emb	1e-3	0.00	32	0.0	10
Image	Mamba	116K	64	3	(32, 2, 4)	N/A	pool	lin	1e-3	0.10	50	1.0	150
Image	MHFA	105K	64	3	4	128	pool	lin	1e-3	0.00	50	0.0	150
Path-32	Mamba	123K	64	2	(64, 2, 4)	64	pool	lin	1e-4	0.05	32	1.0	65
Path-32	MHFA	793K	128	4	8	128	pool	lin	5e-5	0.00	32	0.0	65
Path-64	Mamba	242K	64	4	(64, 2, 4)	64	pool	lin	1e-4	0.05	32	0.0	100
Path-64	MHFA	598K	128	6	8	128	pool	lin	5e-5	0.00	32	0.0	N/A

Table 3: Configuration of the best-performing Mamba/MHFA-based models for different LRA tasks. Here, dim corresponds to the feature dimension, L to the number of layers, h to the number of MHFA-heads, N to Mamba's hidden state dimension, $E = 2$ to the expansion factor, and $d = 4$ to the Conv1D kernel size. Furthermore, FF denotes the size of the FF layer (if it exists), Pool and Enc provide the decoding/encoding mode, LR is the learning rate for the AdamW optimizer, WD is the weight decay, BS is the batch size, and GC the gradient clip values.

Task	Model	Params	dim	Layers	FF	LR	WD	BS	GC
CIFAR	Mamba (real)	153K	64	3	128	0.001	0.1	50	1.0
CIFAR	Mamba (complex)	159K	64	3	128	0.001	0.1	50	1.0
Path-32	Mamba (real)	123K	64	2	64	0.0001	0.05	32	0.0
Path-32	Mamba (complex)	139K	64	2	64	0.0001	0.05	32	0.0

Table 4: Model configurations for the comparison between Mamba models with complex and real A.

which leads to a second-order differential equation. In relatively simple cases, this can be rewritten as a first-order ODE in phase space.[13] Some algebra reveals the simple structure, which we can rewrite in more standard state space notation as[14]

$$\dot{x}(t) = Ax(t) + Bu(t)\,, \tag{7}$$

$$y(t) = Cx(t) + Du(t)\,. \tag{8}$$

SSMs are based on the above equations (7) and (8) and try to approximate is solutions by discretizing the time steps. They are foundational scientific models used in many different scientific fields. The underlying classical mechanics motivation is a particular reason why they are widespread across various fields. In deep learning, early work on SSMs emphasizes their power in capturing long-range dependencies [14, 13]. SSMs are somewhat similar to RNNs and LSTMs but crucially allow for efficient and parallel training and inference at the cost of lacking the versatility of the types of functions they can represent.

[13] For the interested reader: Phase space in physics is the space of position and conjugate momentum (roughly the derivative of the position vector), and the "Hamiltonian" governs the dynamics of the system", and "Hamilton's equations." This formalism is fundamental as it is straightforwardly adapted for quantum mechanics, where we "quantize" the system and deform the Poisson algebra on the phase space.

[14] While in general the input and outputs can be multi-dimensional, for the sake of our exposition here, we shall restrict to the simple 1-dimensional case. Furthermore, we usually set $D = 0$ in deep learning applications, as it is simply a skip connection.

B.1 Discretization

To deal with (7) and (8) in a deep learning setting, we have to discretize the time steps in order to deal with inputs of the form $(x_0, \dots, x_n)$. We can view the discrete input as sampling from a continuous $x(t)$, *i.e.* $x_k = x(k\Delta)$. To discretize, one can introduce a step size Δ, representing the input's resolution. Then, to discretize the continuous-time SSM, S4 uses Tustin's method [33], which maps the state matrix A into an approximation $\bar{A}$:[15]

$$h_k = \bar{A} h_{k-1} + \bar{B} x_k \,, \qquad \bar{A} = (I - \Delta/2A)^{-1} (I + \Delta/2A) \,, \tag{9}$$

$$y_k = C h_k \,, \qquad \bar{B} = (I - \Delta/2A)^{-1} \Delta B \,. \tag{10}$$

Tustin's method – also known as the bilinear method – is a Möbius transformation, and there are various reasons it is often used in signal processing to map continuous-time systems into discrete ones. Notice that in the Mamba implementation, the authors choose a different discretization scheme, called "zero-order hold," which describes the effect of converting a continuous signal into a discrete one by holding each sample value for a certain interval.

We notice that in (11) and (10), we are now mapping $x_k \mapsto y_k$, and like RNNs, h_k can be considered a hidden state with linear transition matrix $\bar{A}$.

B.2 Convolution

The main power of SSMs is due to their ability to be parallelized via a convolution. This heavily relies on the matrices A, B, and C being constant across inputs, limiting the function complexity such models can mimic. Intuitively, the RNN in equations (11) and (10) can be turned into CNNs by unrolling the recursion. Setting $h_{-1} = 0$, we get:

$$h_0 = \bar{B} x_k \,, \quad h_1 = \bar{A}\bar{B} x_0 + \bar{B} x_1 \,, \quad h_2 = \bar{A}^2 \bar{B} x_0 + \bar{A}\bar{B} x_1 + \bar{B} x_2 \,, \quad \dots \,, \tag{11}$$

$$y_0 = \bar{C}\bar{B} h_k \,, \quad y_1 = \bar{C}\bar{A}\bar{B} h_0 + \bar{C}\bar{B} h_1 \,, \quad y_2 = \bar{C}\bar{A}^2\bar{B} h_0 + \bar{C}\bar{A}\bar{B} h_1 + \bar{C}\bar{B} h_2 \,, \quad \dots \,, \tag{12}$$

which can conveniently be vectorized as follows

$$\vec{y} = \mathcal{K} \star \vec{x} \,, \qquad \mathcal{K} = \left(\bar{C}\bar{B}, \bar{C}\bar{A}\bar{B}, \dots, \bar{C}\bar{A}^{L-1}\bar{B} \right) \in \mathbb{R}^L \,, \tag{13}$$

where $\mathcal{K}$ is the convolutional kernel/filter of the SSM. This is, of course, a large convolution over the whole sequence. However, it is well-known that using (fast) Fourier transform (FFT), convolutions in position space can be mapped to simple multiplications in frequency space (and vice-versa). Therefore, we can compute the above convolution as an FFT combined with a multiplication and an inverse FFT, which leads to a $\mathcal{O}(N \log N)$ algorithm. Notice however, that this still requires $\mathcal{O}(N^2 L)$ operations and $\mathcal{O}(NL)$ memory. Therefore, while fast, S4 implements a way to speed this up by carefully pre-conditioning $\bar{A}$ so that we can directly access its spectrum instead of computing powers of $\bar{A}$. Since this is less relevant for Mamba, we refer to the original paper for more details.

C The Selective Scan

In this Appendix, we shall briefly outline the selective scan, which is one of the significant innovations in Mamba. The selective scan allows for parallelizing Mamba's training, thus optimizing it for modern GPU-based training. The idea is based on Blelloch's prefix sum algorithm, which allows for computing the prefix sum of a sequence of numbers in parallel [4]. It was applied to RNNs in [21], and later to SSMs in the S5 paper [29]. Finally, Mamba exploited it for their input-dependent/selective approach and further improved the performance using a hardware-aware CUDA implementation, another crucial ingredient in making their model competitive with Transformers.

The Blelloch prefix sum algorithm computes the prefix sum by first doing a parallel up-sweep on a binary tree followed by a down-sweep (see Figure 7 for an example of how this works). Little care has to be taken when the number of input elements is not of size 2^d, and we refer to the original paper for more details [4].

Crucially, the Blelloch scan for a recurrence of the type $h_t = (\Lambda_t \otimes h_{t-1}) \oplus x_t$ only requires the operation, $\oplus$ and $\otimes$, which in the example of Figure 7 are just addition and multiplication by 1, to have the following properties:

[15] We change notation here compared with (7) and (8) in order to be compatible with the main text.

(i) $\oplus$ is associative,

(ii) there exists a binary associative operator $\odot$ such that $x \otimes (y \otimes z) = (x \odot y) \otimes z$, and

(iii) $x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$.

For example, vector addition $x + y$ together with Ax and AB, vector-matrix and matrix-matrix multiplication, respectively, satisfy these conditions.

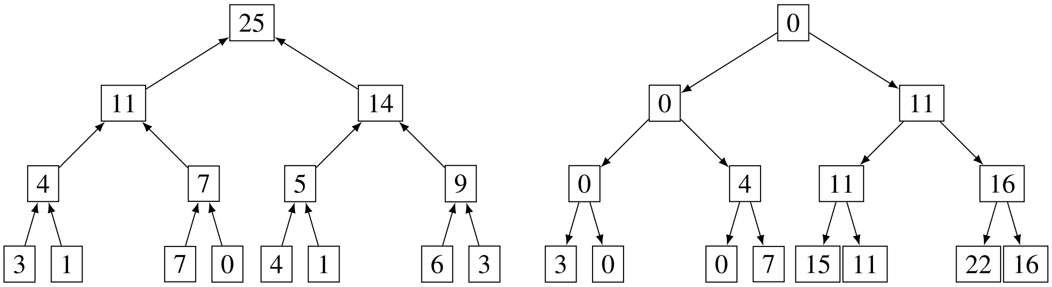

Figure 7: Blelloch scan for the example input of $[3, 1, 7, 0, 4, 1, 6, 3]$. We depict the up-sweep operation on the left and the corresponding down-sweep on the right. This example shows how the Blelloch scan allows for parallelization and that the time complexity for computing the prefix sum can be reduced to $\mathcal{O}\left(n/p + \log p\right)$ upon parallelizing on p processors.

We are mostly interested in how to apply this to the Mamba/SSM case. Notice that once we have the matrices $\bar{A}_t$ and $\bar{B}_t$ computed for each input x_t, these two cases are equivalent from the perspective of the scan operation. Recall that we have to compute the following system of equations:

$$h_t = \bar{A}_t h_{t-1} + \bar{B}_t x_t \,, \tag{14}$$

$$y_t = C_t h_t \,. \tag{15}$$

We are interested in parallelizing the computation of h_t. To do so, we can unravel the recurrence to obtain

$$\begin{aligned}
h_1 &= \bar{B}_1 x_1 \,, \\
h_2 &= \bar{A}_2 \bar{B}_1 x_1 + \bar{B}_2 x_2 \,, \\
h_3 &= \bar{A}_3 \bar{A}_2 \bar{B}_1 x_1 + \bar{A}_3 \bar{B}_2 x_2 + \bar{B}_3 x_3 \,, \\
h_4 &= \bar{A}_4 \bar{A}_3 \bar{A}_2 \bar{B}_1 x_1 + \bar{A}_4 \bar{A}_3 \bar{B}_2 x_2 + \bar{A}_4 \bar{B}_3 x_3 + \bar{B}_4 x_4 \,. \\
&\cdots \,.
\end{aligned}$$

To define operators that satisfy conditions (i) – (iii) for the Blelloch scan, we introduce the tuple γ_t together with the operator $\star$:

$$\gamma_t = \left(\bar{A}_t, \bar{B}_t x_t\right) \,, \qquad \gamma_i \star \gamma_j = \left(\bar{A}_j \bar{A}_i, \bar{A}_j \bar{B}_i x_i + \bar{B}_i x_i\right) \,. \tag{16}$$

It is then straightforward to show that this matrix-vector tuple together with the $\star$ operation satisfies the properties (i) – (iii), and therefore allows for parallel evaluation according to the Blelloch scan (see Appendix H of [29]).