CSE 291 (L00) Project Report: Mamba State Space Model

Sanidhya Singal (`ssingal@ucsd.edu`) and Aditya Gulati (`adgulati@ucsd.edu`)
Department of Computer Science and Engineering
University of California San Diego

Winter 2024

1 Introduction

Deep learning applications have seen substantial advancements with the advent of the Transformer architecture and its attention mechanism [14]. Despite its success, Transformers face challenges in handling long sequences due to their inherent computational inefficiency. Several subquadratic-time architectures [13] have been proposed to address this limitation (refer to figure 1), but they have not matched attention-based models' performance on crucial tasks such as language processing [12]. In this report, we identify the inability of these models to perform content-based reasoning as a key weakness and focus on Mamba [3], a novel neural network architecture that integrates selective structured state space models (SSMs) to address this limitation.

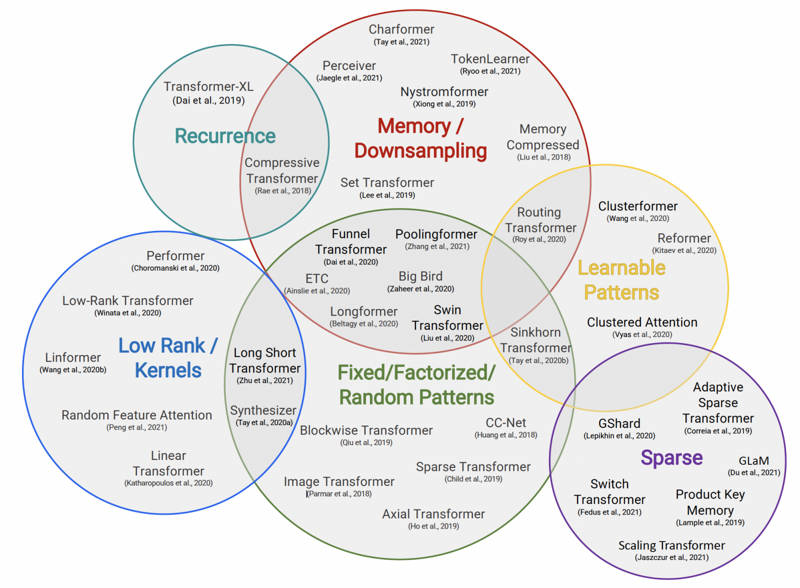

Charformer (Tay et al., 2021)
TokenLearner (Ryoo et al., 2021)
Perceiver (Jaegle et al., 2021)
Nystromformer (Xiong et al., 2019)
Transformer-XL (Dai et al., 2019)
Recurrence
Memory / Downsampling
Memory Compressed (Liu et al., 2018)
Compressive Transformer (Rae et al., 2018)
Set Transformer (Lee et al., 2019)
Clusterformer (Wang et al., 2020)
Routing Transformer (Roy et al., 2020)
Reformer (Kitaev et al., 2020)
Funnel Transformer (Dai et al., 2020)
Poolingformer (Zhang et al., 2021)
Performer (Choromanski et al., 2020)
Learnable Patterns
ETC (Ainslie et al., 2020)
Big Bird (Zaheer et al., 2020)
Low-Rank Transformer (Winata et al., 2020)
Longformer (Beltagy et al., 2020)
Swin Transformer (Liu et al., 2020)
Clustered Attention (Vyas et al., 2020)
Sinkhorn Transformer (Tay et al., 2020b)
Linformer (Wang et al., 2020b)
Low Rank / Kernels
Long Short Transformer (Zhu et al., 2021)
Fixed/Factorized/ Random Patterns
Adaptive Sparse Transformer (Correia et al., 2019)
Synthesizer (Tay et al., 2020a)
Random Feature Attention (Peng et al., 2021)
CC-Net (Huang et al., 2018)
GShard (Lepikhin et al., 2020)
Blockwise Transformer (Qiu et al., 2019)
Sparse
GLaM (Du et al., 2021)
Linear Transformer (Katharopoulos et al., 2020)
Sparse Transformer (Child et al., 2019)
Image Transformer (Parmar et al., 2018)
Switch Transformer (Fedus et al., 2021)
Product Key Memory (Lample et al., 2019)
Axial Transformer (Ho et al., 2019)
Scaling Transformer (Jaszczur et al., 2021)

Figure 1: Taxonomy of Efficient Transformer Architectures (Image Source: [13])

Mamba [3] introduces content-based reasoning by allowing SSM parameters to be functions of the input, enabling the model to selectively propagate or forget information along the sequence length dimension based on the current token. While this change prevents the use of efficient convolutions, the authors design a hardware-aware parallel algorithm in recurrent mode to address the computational efficiency issue. They integrate selective SSMs into an end-to-end neural network architecture without attention or MLP blocks.

Mamba achieves fast inference (5× higher throughput than Transformers) and linear scaling in sequence length, outperforming Transformers of comparable size in language modeling. The Mamba-3B model matches the performance of Transformers twice its size in both pretraining and downstream evaluation. Furthermore, Mamba achieves state-of-the-art performance across several modalities including audio and genomics, demonstrating its versatility and effectiveness as a general sequence model backbone.

2 Related Work

Long Range Dependencies in State Space Models

The advancement of deep generative models has enabled the development of complex applications across various domains. However, the computational efficiency of these models remains a significant challenge, particularly when dealing with long sequences. Recurrent Neural Networks (RNNs), known for their compact state representation, struggle with sequences beyond approximately 100 steps. Conversely, the Transformer architecture, powered by self-attention mechanisms, has been the gold standard in terms of performance but scales quadratically with sequence length.

One notable effort to address this issue is the **Long-Range Arena (LRA)** benchmark introduced in [12]. The LRA benchmark evaluates model quality under long-context scenarios using tasks such as Long List Ops and Pathfinder. In the former task, the model is given an input consisting of a long list of nested operators and needs to calculate an answer, while the latter is a fun task where the model gets a flattened image (refer to figure 2) and needs to determine whether the two circles can be joined. The authors found that it is challenging to improve significantly upon the performance of vanilla Transformers while maintaining the same benchmark score (refer to figure 3).

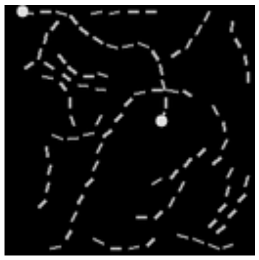


Figure 2: The Pathfinder Task in the LRA Benchmark (Image Source: [12])

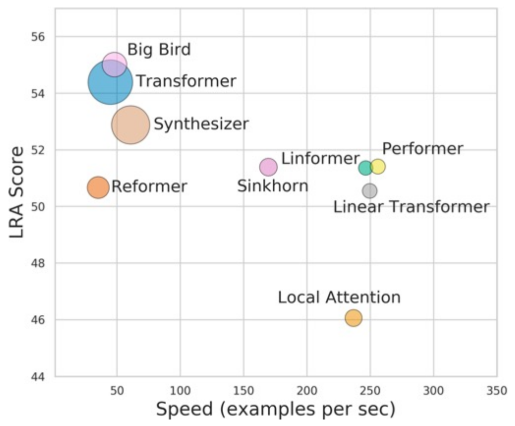


Figure 3: LRA Score for Various Models (Image Source: [12])

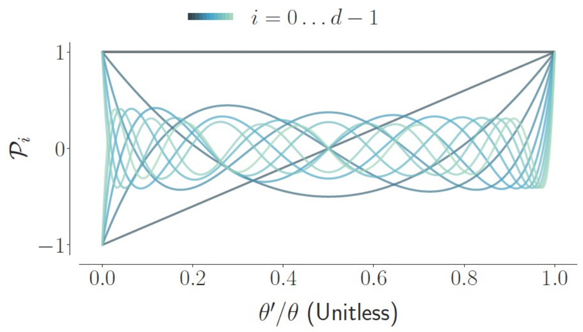


Figure 4: Legendre Polynomials (Image Source: [15])

In [15], the authors propose **Legendre Memory Units** based on **Legendre Polynomials** (refer to figure 4), which are constructed by a sequence of polynomials that are orthogonal to each other. They show that we can store the history of our system by projecting the input values onto a basis of these polynomials.

Building on this idea, **HiPPO** [4] phrases memory as a technical problem of online function approximation where a function $f(t) : \mathbb{R}_+ \to \mathbb{R}$ is summarized by storing its optimal coefficients in terms of some basis functions. Using this insight, the authors construct a simple model based on Legendre polynomials.

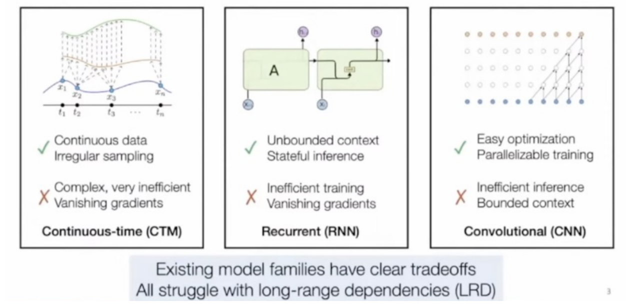


Figure 5: Paradigms for Long Range Dependencies (Image Source: [4])

In [6], Gu et al. compares and contrasts Continuous Time Models (CTMs), Recurrent Neural Networks (RNNs), and Convolutional Neural Networks (CNNs) for their inabilities to address long-range dependencies (refer to figure 5). Specifically, CTMs try to model the underlying continuous process of data, which allows them to capture the inductive bias of this type of data better and perhaps automatically do things like handling missing data. And RNNs are natural stateful models where they summarize information they've seen to a hidden state and so they're better in certain stateful settings like reinforcement learning and auto-regressive tasks. However, both CTMs and RNNs are slow and suffer from the vanishing gradients problem because they're inherently sequential. On the other hand, CNNs are easily parallelizable, so they scale much better and are much easier to train. However, they have an inherently bounded context which makes them unable to address long dependencies.

In an attempt to combine the strengths of different sequence modeling paradigms, the work on **Linear State Space Layers (LSSL)** [6] proposed a unified framework based on a standard state space representation. Given an input u and a hidden state x (of higher dimension than u), the output y (of same dimension as u) can be modeled as a linear function of u and x as follows:

$$x'(t) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t) + Du(t)$$

This is a continuous-time state space representation. The corresponding discrete-time state space representation is given by:

$$x_t = \bar{A}x_{t-1} + \bar{B}u_t$$

$$y_t = Cx_t + Du_t$$

In other words, this work showed that by starting from an implicit continuous time state space model, one could discretize to produce a system that can be interpreted as a recurrent network with the benefit of efficient inference or as a convolutional model which benefits from parallelizable training (refer to figure 6).

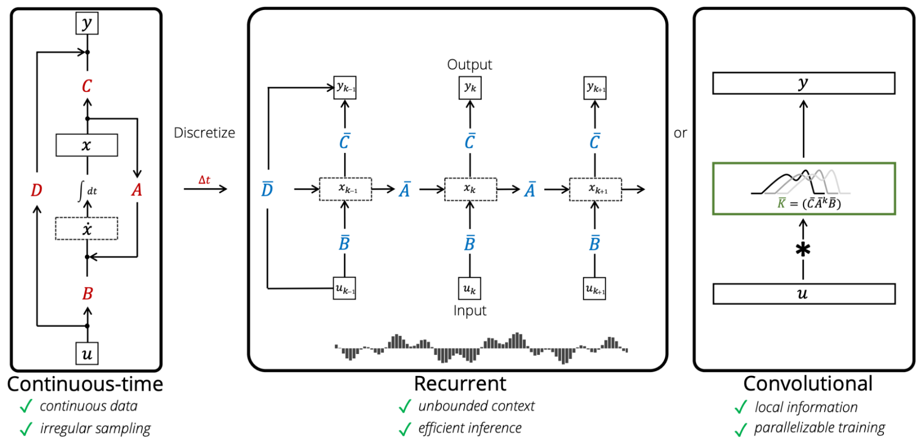


Figure 6: (Left) As an implicit continuous model, irregularly-spaced data can be handled by discretizing the same matrix A using a different timescale Δt. (Center) As a recurrent model, inference can be performed efficiently by computing the layer timewise, i.e., one vertical slice at a time by unrolling the linear recurrence. (Right) As a convolutional model, training can be performed efficiently by computing the layer depthwise in parallel by convolving with a particular filter (Image Source: [6]).

However, a naive implementation of these models requires a massive amount of memory. In fact, for reasonably sized models, such as when the state dimension is 256, LSSL uses orders of magnitude more memory than comparably sized RNNs or CNNs.

Efficiently Modeling Long Sequences with Structured State Spaces (S4) [5] builds upon these previous efforts by proposing a novel architecture that combines the advantages of convolutional interpretations with state space models. Specifically, to train a SSM using the convolutional representation, the authors preform unrolling and vectorization to get a L-dimensional kernel ($\bar{K}$) where L is the sequence length. The details are shown in figures 7 and 8.

The scary part here is the matrix A raised to the power of i, where $i \in [0, L-1]$. This proposes several problems:

$$x_0 = \overline{\boldsymbol{B}}u_0 \qquad x_1 = \overline{\boldsymbol{AB}}u_0 + \overline{\boldsymbol{B}}u_1 \qquad x_2 = \overline{\boldsymbol{A}}^2\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{AB}}u_1 + \overline{\boldsymbol{B}}u_2 \qquad \dots$$

$$y_0 = \overline{\boldsymbol{CB}}u_0 \qquad y_1 = \overline{\boldsymbol{CAB}}u_0 + \overline{\boldsymbol{CB}}u_1 \qquad y_2 = \overline{\boldsymbol{CA}}^2\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{CAB}}u_1 + \overline{\boldsymbol{CB}}u_2 \qquad \dots$$

Figure 7: Unrolling the Discrete-time State Space Representation (Image Source: [5])

$$y_k = \overline{\boldsymbol{CA}}^k\overline{\boldsymbol{B}}u_0 + \overline{\boldsymbol{CA}}^{k-1}\overline{\boldsymbol{B}}u_1 + \cdots + \overline{\boldsymbol{CAB}}u_{k-1} + \overline{\boldsymbol{CB}}u_k$$
$$y = \overline{\boldsymbol{K}} * u.$$

$$\overline{\boldsymbol{K}} \in \mathbb{R}^L := \mathcal{K}_L(\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \overline{\boldsymbol{C}}) := \left(\overline{\boldsymbol{CA}}^i\overline{\boldsymbol{B}}\right)_{i\in[L]} = (\overline{\boldsymbol{CB}}, \overline{\boldsymbol{CAB}}, \dots, \overline{\boldsymbol{CA}}^{L-1}\overline{\boldsymbol{B}}).$$

Figure 8: Vectorizing the Unrolled Discrete-time State Space Representation into a Convolution (Image Source: [5])

1. High powers of matrix A can lead to vanishing gradients.
2. High powers of matrix A are expensive to compute and can take $\mathcal{O}(N^2L)$ computations, whereas, ideally, this should take $\mathcal{O}(L)$ computations.

To overcome these challenges, S4 uses HiPPO operators for the A matrix. The authors show that through the application of Inverse FFT and Woodbury Identity, the matrix A can be diagonalized stably and the SSM can be reduced to the well-studied computation of a Cauchy kernel [5]. This overcomes the problem of vanishing gradients and takes only $\mathcal{O}(N + L)$ computations to compute higher powers of A.

Consequently, the S4 model achieves state-of-the-art performance on the LRA benchmark (for sequence lengths between 1k and 16k), even excelling in tasks like Path-X, where the model must determine if two points are connected on a fairly large image. The results are shown in figure 9.

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X	Avg
Transformer	36.37	64.27	57.46	42.44	71.40	✗	53.66
Reformer	<u>37.27</u>	56.10	53.40	38.07	68.50	✗	50.56
BigBird	36.05	64.02	59.29	40.83	74.87	✗	54.17
Linear Trans.	16.13	<u>65.90</u>	53.09	42.34	75.30	✗	50.46
Performer	18.01	65.40	53.82	42.77	77.05	✗	51.18
FNet	35.33	65.11	59.61	38.67	<u>77.80</u>	✗	54.42
Nyströmformer	37.15	65.52	<u>79.56</u>	41.58	70.94	✗	57.46
Luna-256	37.25	64.57	79.29	<u>47.38</u>	77.72	✗	<u>59.37</u>
S4	**59.60**	**86.82**	**90.90**	**88.65**	**94.20**	**96.35**	**86.09**

Figure 9: Comparative results for the LRA benchmark (Image Source: [5])

Finally, we take a look at two of the current state-of-the-art SSMs: Hungry Hungry Hippos (H3) [2], and Hyena Hierarchy [10].

Hungry Hungry Hippos (H3) [2] builds on top of S4 and Linear Attention [7] by adding Shift SSMs and Diagonal SSMs (refer to figure 10).

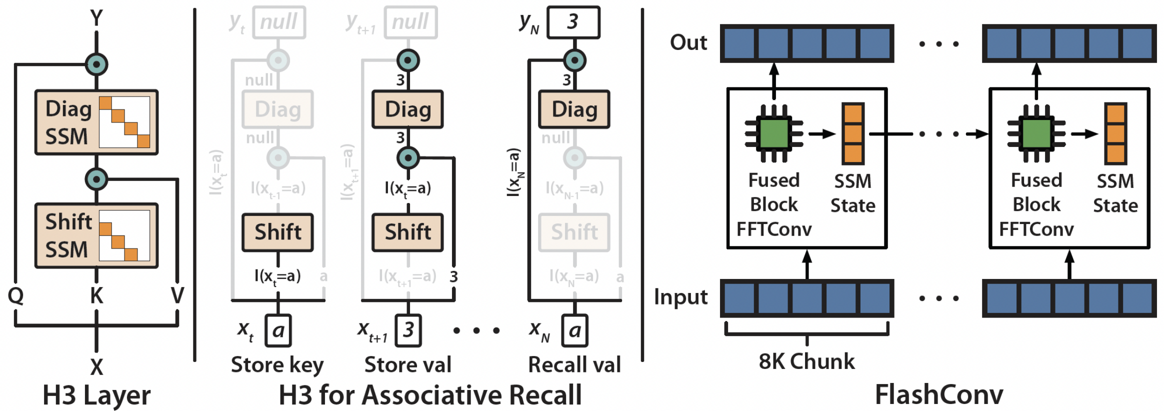


Figure 10: (Left) H3 stacks two discrete SSMs with shift and diagonal matrices and uses multiplicative interactions between input projections and their outputs to model comparisons between points in a sequence. (Middle) H3 can perform associative recall, which is easy for attention, but not existing SSMs. (Right) FlashConv uses a new state-passing algorithm over fused block FFTConv to increase hardware efficiency of SSMs, allowing H3 to scale to billion-parameter models (Image Source: [2]).

- Shift SSM allows the model to capture dependencies between nearby tokens. An example would be to output previous token when provided with a current token.
- Diagonal SSM allows the model to remember state over the entire sequence. This allows the model to behave similar to an attention layer and perform associative recall.

Given an input u, we project it to get three signals $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ similar to Transformers, from which the output is calculated as follows:

$$\mathbf{Q} \odot \text{SSMdiag}(\text{SSMshift}(\mathbf{K}) \odot \mathbf{V})$$

Here, $\odot$ denotes pointwise multiplication. We can view this form as stacking two SSMs with multiplicative interaction (each is called a “hungry hippo”). Furthermore, during training, H3 uses FlashConv which is an FFT-based convolution architecture. This allows for fast training and scaling to billion-parameter models.

Though H3 employs standard local convolution to capture dependencies between nearby tokens, **Hyena Hierarchy** [10] innovates by adding a MLP-parameterized global convolution. This means that instead of having a fixed kernel like S4, we now have kernels that are learnt with the help of an MLP. This allows the model to adapt to different tasks and capture diverse long-range dependencies. More so, gates control the flow of information between states, allowing the model to focus on relevant information.

So, through these innovations, Hyena is able to capture long range dependencies. And particularly due to global convolutions, it is scalable and much more efficient than other state space models. Finally, we can visualize the learnt convolution kernels and find insights about how the model learns and utilizes long-range dependencies.

But how do these models compare with the new Mamba model? The next section discusses the contributions of the Mamba State Space Model.

3 Mamba State Space Model

Mamba [3] introduces several key contributions to the SSM approach, aiming to enhance its efficiency and effectiveness. Firstly, it incorporates a **selection mechanism**, addressing the prior model's inability to select data efficiently in an input-dependent manner. This mechanism is designed to parameterize the state space model parameters based on the input sequence, enabling the model to behave differently based on the input.

However, the introduction of this selection mechanism complicates the use of efficient convolutional tricks seen in previous models like S4 [5]. Therefore, the second major contribution of Mamba is a **hardware-aware algorithm** that computes the model recurrently using a scan instead of convolution. This algorithm leads to a faster implementation compared to previous methods, scaling linearly in sequence length compared to pseudo-linear scaling for all convolution-based state space models.

The third significant contribution of Mamba is the **simplification of prior deep sequence model architectures**. It combines the design of SSM architectures with the MLP block of Transformers into a single block, resulting in a simple and homogeneous architecture design called Mamba.

3.1 Motivation for Selection Mechanism

The core motivation behind Mamba's design is to utilize selection as a means of compression. Different sequence models make different trade-offs when tackling the fundamental problem of sequence modeling, which is compressing context into a smaller state. Attention, for example, doesn't compress context at all. On the other hand, while recurrent models are efficient but they are limited in effectiveness based on their ability to compress context.

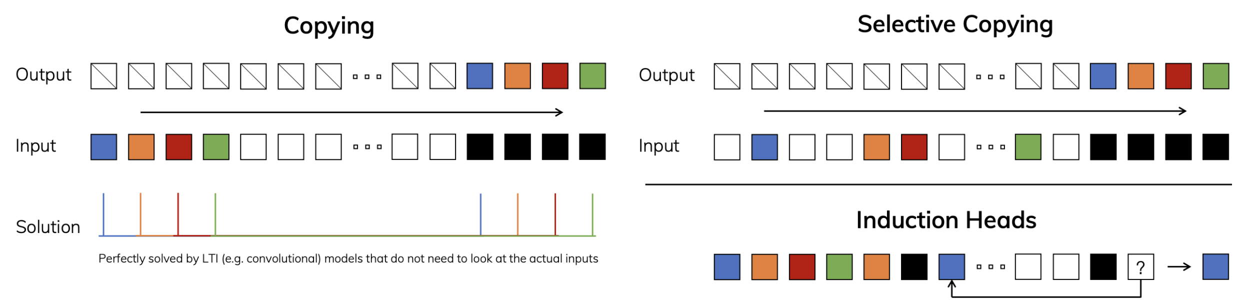
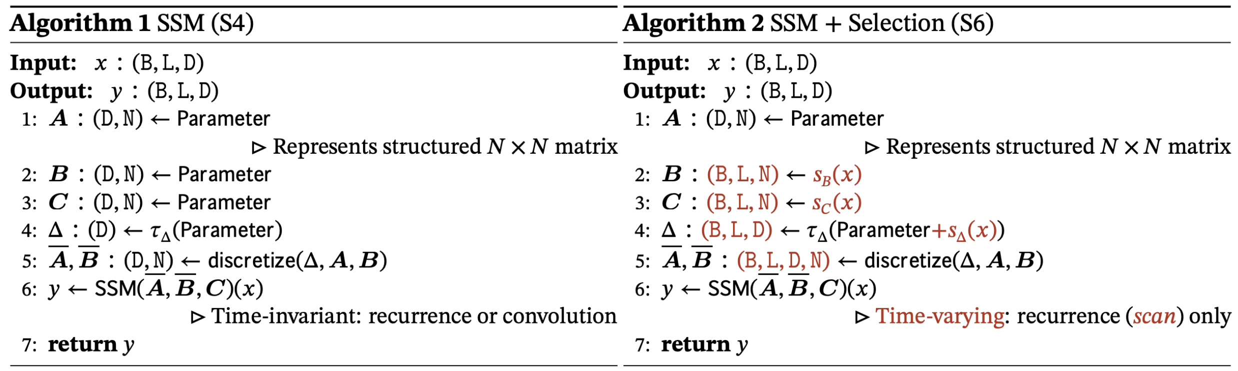

Copying
Output
Input
Solution
Perfectly solved by LTI (e.g. convolutional) models that do not need to look at the actual inputs
Selective Copying
Output
Input
Induction Heads
?

Figure 11: (Left) The Copying task involves constant spacing between input and output elements and is easily solved by time-invariant models such as linear recurrences and global convolutions. (Right Top) The Selective Copying task has random spacing in between inputs and requires time-varying models that can selectively remember or ignore inputs depending on their content. (Right Bottom) The Induction Heads task is an example of associative recall that requires retrieving an answer based on context, a key ability for LLMs (Image Source: [3]).

To explore this trade-off, the authors focus on three synthetic tasks. As shown in figure 11, the job of the model in the "copying" task is to copy color sequences in the input to the output, after being delayed by some constant offset. Standard convolutional models with fixed kernels can solve this without any problem. The first of the harder tasks is "selective copying". Here, we have random spaces symbolized by the white blocks in between the input sequence elements. To successfully copy the color blocks to the output while ignoring the white blocks, we need a mechanism that behaves differently at different inputs. The second hard task is “induction heads". Here, in order to predict the appropriate color at the question mark block, the model needs to be able to perform retrieval back over the input sequence (i.e., use the context provided by the black square to predict that blue is the next color in the sequence).



Algorithm 1 SSM (S4)

```
Input: $x$ : (B, L, D)
Output: $y$ : (B, L, D)
1: $\boldsymbol{A}$ : (D, N) ← Parameter
                              ▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B}$ : (D, N) ← Parameter
3: $\boldsymbol{C}$ : (D, N) ← Parameter
4: $\Delta$ : (D) ← $\tau_\Delta$(Parameter)
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}$ : (D, N) ← discretize($\Delta, \boldsymbol{A}, \boldsymbol{B}$)
6: $y$ ← SSM($\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C}$)($x$)
                              ▷ Time-invariant: recurrence or convolution
7: return $y$
```

Algorithm 2 SSM + Selection (S6)

```
Input: $x$ : (B, L, D)
Output: $y$ : (B, L, D)
1: $\boldsymbol{A}$ : (D, N) ← Parameter
                              ▷ Represents structured $N \times N$ matrix
2: $\boldsymbol{B}$ : (B, L, N) ← $s_B(x)$
3: $\boldsymbol{C}$ : (B, L, N) ← $s_C(x)$
4: $\Delta$ : (B, L, D) ← $\tau_\Delta$(Parameter+$s_\Delta(x)$)
5: $\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}$ : (B, L, D, N) ← discretize($\Delta, \boldsymbol{A}, \boldsymbol{B}$)
6: $y$ ← SSM($\overline{\boldsymbol{A}}, \overline{\boldsymbol{B}}, \boldsymbol{C}$)($x$)
                              ▷ Time-varying: recurrence (scan) only
7: return $y$
```

Figure 12: Side-by-side comparison of SSM (S4) and SSM + Selection (S6) Algorithms (Image Source: [3])

Mamba builds upon the S4 state space model [5], which utilizes parameters that don’t depend on the input sequence. However, to allow the model to behave differently on different inputs, the authors modify B, C, and Δ to be time-varying and therefore, dependent on the input sequence x (refer to figure 12). This modification increases the model’s power but also makes it impossible to use the efficient S4 convolution trick.

3.2 Motivation for Hardware-Aware State Expansion

In effect, the two big challenges to be tackled once we give up on convolution are the sequential nature of recurrence and the large memory usage. A key idea for getting around the sequential processing problem comes from **Simplified State Space Layers for Sequence Modeling (S5)** [11] which highlights the benefits of using parallel scans to maintain efficiency while avoiding the use of convolutional tricks used in S4. Mamba uses this parallel scan idea in combination with efficient use of memory.

Figure 13 shows the Selective State Space Model with Hardware-Aware State Expansion, which illustrates the incorporation of a selection mechanism and the architecture’s loading and update processes. Here are a few things to note:

1. B_t, Δ_t and C_t all depend on the input x_t via the Selection Mechanism.
2. The model parameters are loaded into fast GPU SRAM where the updates are performed (shown in orange).
3. The output is ultimately stored back into GPU HBM (shown in green).

In other words, instead of preparing the scan input $(\bar{A}, \bar{B})$ of size $B \times L \times D \times N$ in GPU high bandwidth memory (HBM), the authors load the SSM parameters A, B, C, Δ directly from the slow HBM to fast SRAM, where they perform the discretization and recurrence. Then, they write the final results of size $B \times L \times D$

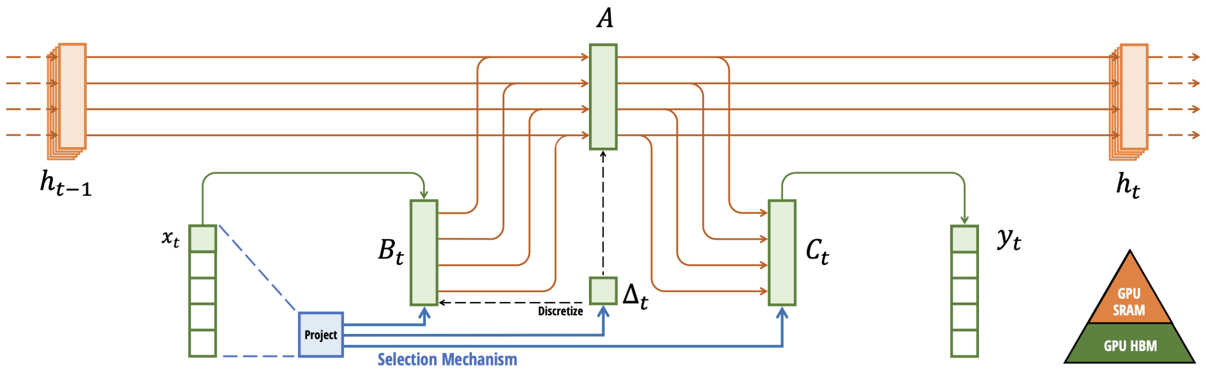


Figure 13: Selective State Space Model with Hardware-Aware State Expansion (Image Source: [3])

back to the HBM.

Lastly, the authors use recomputation to reduce the memory requirements. This allows them to avoid saving the intermediate states, which are only necessary for back propagation.

3.3 Simplified SSM Architecture

Finally, the authors propose a simplified SSM architecture that combines the Hungry Hungry Hippos (H3) block [2] with a gated MLP block (as used in almost every Transformer architecture [14]) to form the Mamba block (refer to figure 14). This block is repeated and interleaved with standard normalization and residual connections to create the Mamba architecture.

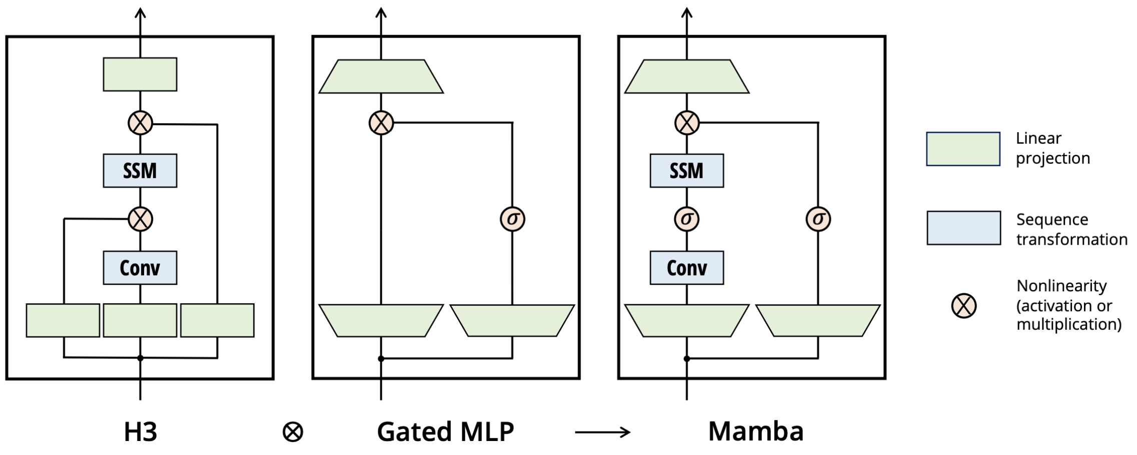


Figure 14: Mamba Architecture based on the H3 Block and the Gated MLP Block (Image Source: [3])

The authors make the following notes:

1. Compared to the H3 block, Mamba replaces the first multiplicative gate with an activation function.
2. Compared to the MLP block, Mamba adds an SSM to the main branch.

3. For σ, the authors use the SiLU or Swish activation. These are given by:

$$\text{SiLU}(x) = x * \text{sigmoid}(x)$$

$$\text{Swish}_\beta(x) = x * \text{sigmoid}(\beta x)$$

where x is the input and β is a learnable parameter. Clearly, $\text{Swish}_1(x) = \text{SiLU}(x)$.

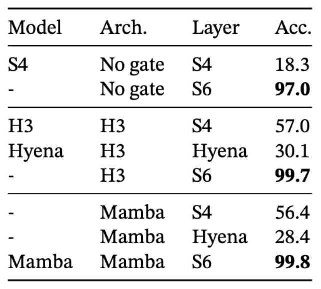
Finally, the authors note that most prior state space models use complex numbers in their state, but it has been empirically observed that completely real-valued state space models may work as well or better in some settings. Therefore, Mamba defaults to using real values, which have been effective in most tasks.

The authors call these Selective SSMs as S6 models, because they are S4 models with a 'selection' mechanism and computed with a parallel 'scan' using hardware-aware state expansion.

3.4 Discussion

Following are the evaluation details along with results and inferences:

1. Experiments on Synthetic Tasks such as Copying, Selective Copying, and Induction Heads: S6 works well with every architecture but S4 and Hyena work comparatively poorly on the induction heads task. Mamba succeeds on up to million-length sequences, which are 4000× longer than it saw during training, while no other method goes beyond 2× the training length.



Model	Arch.	Layer	Acc.
S4	No gate	S4	18.3
-	No gate	S6	**97.0**
H3	H3	S4	57.0
Hyena	H3	Hyena	30.1
-	H3	S6	**99.7**
-	Mamba	S4	56.4
-	Mamba	Hyena	28.4
Mamba	Mamba	S6	**99.8**

Table 1: (**Selective Copying**.) Accuracy for combinations of architectures and inner sequence layers.

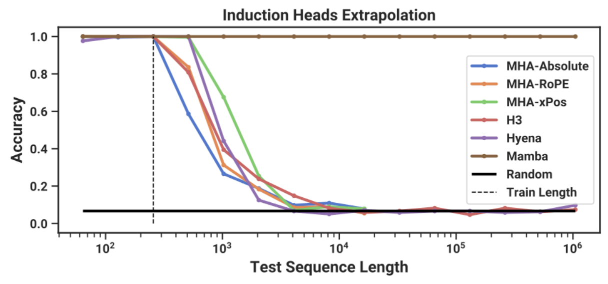


Table 2: (**Induction Heads**.) Models are trained on sequence length $2^8 = 256$, and tested on increasing sequence lengths of $2^6 = 64$ up to $2^{20} = 1048576$. Full numbers in Table 11.

Figure 15: Mamba Performance on Vanilla Tasks (Image Source: [3])

Figure 15 showcases the results on the vanilla tasks of Selective Copying and Induction heads. The main motivation for selective copying task is to study the data-dependence aspect of language models which comes inherently by variable spacing over the copying task. The Induction head task replicates the in-context learning ability of LLMs. It requires models to perform associative recall and copy. For example, if the model has seen a bigram such as "Harry Potter" in the sequence, then when the next time "Harry" appears in the same sequence, the model should be able to predict "Potter" by copying from history.

2. Experiments on Language Modeling:
   (a) Scaling Laws: Language modeling up to 1.3B parameters matches that of an attention transformer using a PaLM- and LLaMA-like training recipe.
   (b) Zero-Shot Downstream Evaluations: Mamba models with up to 2.8B parameters win against Pythia and other models with similar parameter counts on standard benchmarks.

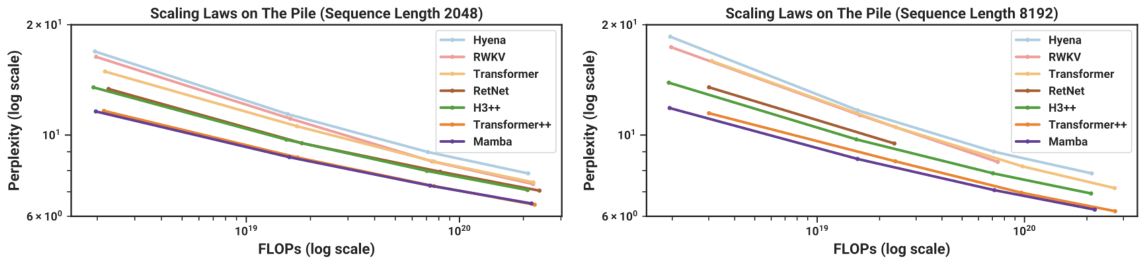


Figure 16: (Scaling Laws) Models of size 125M to 1.3B parameters, trained on the Pile. Mamba scales better than all other attention-free models and is the first to match the performance of a very strong "Transformer++" recipe that has now become standard, particularly as the sequence length grows (Image Source: [3]).

Figure 16 highlights the scaling laws under the standard Chinchilla protocol on models from 125M to 1.3B parameters. Mamba is the only attention-free model to match the performance of Transformer (Transformer++) as the sequence length grows.

3. Experiments on DNA Modeling: Mamba's pretraining perplexity improves smoothly with model size, and that Mamba scales better than both HyenaDNA and Transformer++.
4. Ablation Studies:
   (a) Architecture: The Mamba block performs similarly to H3 while being simpler.
   (b) Selective SSM: Δ is the most important parameter due to its connection to RNN gating. It balances between focusing on the past and ignoring the current input, or vice versa.
5. Speed and Memory Benchmarks: The authors benchmark the speed of the SSM scan operation as well as the end-to-end inference throughput of Mamba and show that Mamba achieves 5× higher throughput than a Transformer of similar size.

   The bench-marking of the end-to-end inference throughput of Mamba is shown in figure 17. The hardware-aware efficient SSM scan is faster than the best attention implementation beyond sequence lengths of 2k and $20 - 40\times$ faster than standard scan implemented in PyTorch. Mamba also achieves $4 - 5\times$ higher inference throughput than Transformer of similar size. As per the scaling, a Mamba-6.9B (future work) is expected to have higher inference throughput than a 5× smaller Transformer-1.3B, which seems as a promising result.

Despite all its improvements and innovations, Mamba suffers from a few limitations:

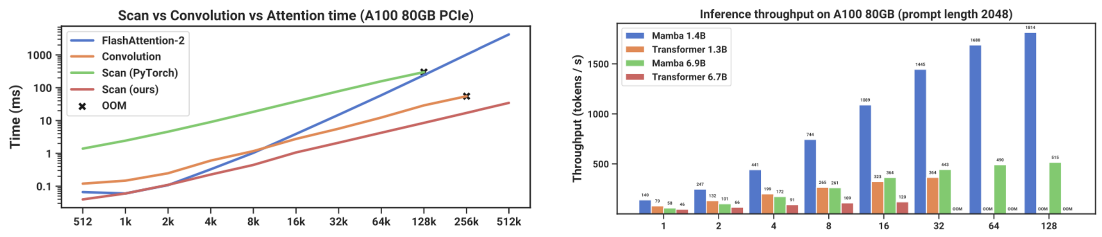


Figure 17: (Efficiency Benchmarks) (Left) Training: Efficient scan is 40× faster than a standard implementation. (Right) Inference: As a recurrent model, Mamba can achieve 5× higher throughput than Transformers (Image Source: [3]).

1. The authors highlight that there is **no free lunch on the Continuous-Discrete Spectrum**. The Selection Mechanism overcomes the weaknesses of prior State Space Models on discrete modalities such as text and DNA, but this can impede their performance on data that Linear Time Invariant State Space Models excel on.

2. They also note that the empirical evaluation is limited to **small model sizes**, below the threshold of most strong open source LLMs. Therefore, it remains to assess whether Mamba still compares favorably at these larger sizes.

In conclusion, Selective State Space Models and their extension, the Mamba architecture, are recurrent models that possess fundamental attributes making them ideal as the foundational models for general sequence-based applications. These models demonstrate the following key properties:

- **High Quality**: Selective SSMs, enabled by the design of the Mamba architecture, exhibit superior performance on dense modalities such as language and genomics. This selectivity allows the model to focus on relevant information, leading to more accurate predictions.

- **Fast Training and Inference**: The computation and memory requirements of Selective SSMs scale linearly with the sequence length during training. Additionally, during inference, the model can be unrolled autoregressively, requiring only constant time per step since it does not need to cache previous elements.

- **Long Context**: The combination of high-quality predictions and efficient computation means that Selective SSMs can effectively handle long sequences. This efficiency results in performance improvements on real data up to sequence lengths of 1 million tokens.

3.5 Current Work

There are many extensions of Mamba's work in terms of research and production. The main highlights of these extensions come from Multi-modal (Images/Graphs), Mixture-of-Experts, and Instruction Fine Tuning use cases:

1. **Mamba-ND** [8], is a multi-dimensional extension of Mamba models for images and videos modalities. The main idea here is to efficiently process information through SSMs via linear scaling in data size. This is achieved by data rearrangement and enhanced bi-directional processing techniques ensuring high performance without hefty computational cost (refer to figures 18 and 19).

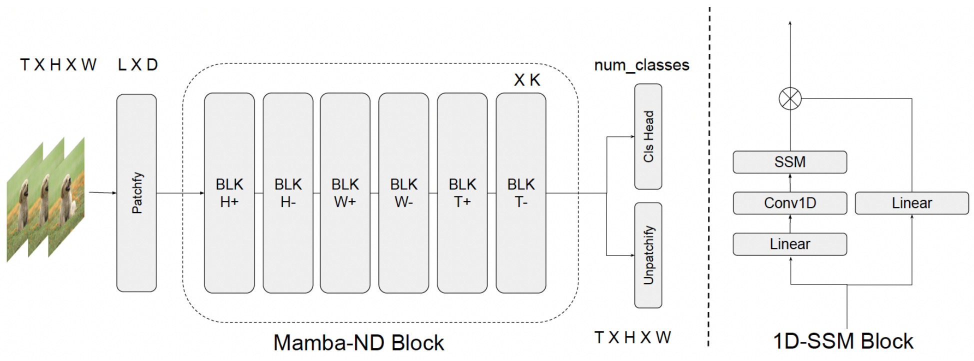


Figure 18: Architecture of Mamba-3D: Given 3D input data, the authors convert it into L patches, while maintaining the original 3D structure of the input. This sequence is then passed through K Mamba-ND blocks, each of which consists of a chain of 1-D Mamba layers that process the sequence in alternating orderings. In 3D space, the order H+H-W+W-T+T-. Finally, the sequence is reshaped back to its original 3D structure and passed to task-specific heads for downstream tasks (Image Source: [8]).

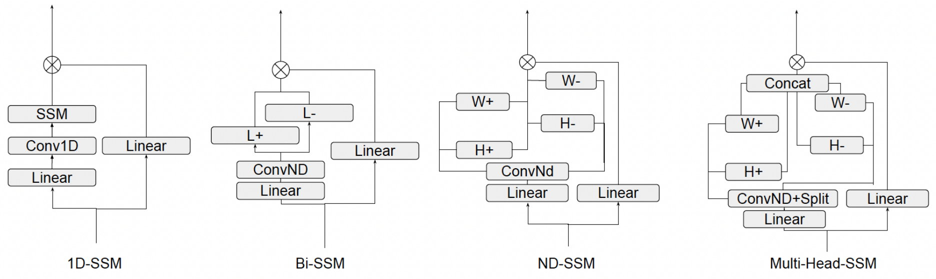


Figure 19: Variations of SSM Layer Design: Col 1 represents the standard 1D SSM layer. Col 2 represents Bi-SSM, which adds bidirectionality in a similar fashion as LSTM. Col 3 represents ND-SSM block, which extends Bi-SSM to more directions. Col 4 represents multi-head SSM block inspired by multi-head attention in Transformers (Image Source: [8]).

2. **Graph-Mamba** [16] is an extension that models relationships and dependencies in graph data. Graphs provide a wide range of applications like DNA sequencing, social graphs, and biological networks. Traditional models fail to model this relationship efficiently. Graph-Mamba addresses this by selectively focusing on relevant nodes (filtering) in a graph, thereby reducing computational demands while capturing intricate and long-range dependencies within the data (refer to figure 20).

3. **BlackMamba** [1] combines the strengths of Mamba with Mixture-of-Experts (MoE) models, offering a best-of-both-worlds equivalent in terms of performance and efficiency (refer to figure 21). By integrating Mamba's linear complexity in handling sequences with the MoE's ability to process information rapidly and accurately, BlackMamba sets a new standard for model efficiency and effectiveness.

4. The aforementioned extensions are based on research settings. In production level settings, NVIDIA AI Foundation Models and Endpoints [9] provides access to a curated set of community and NVIDIA-built generative AI models to experience, customize, and deploy in enterprise applications. The

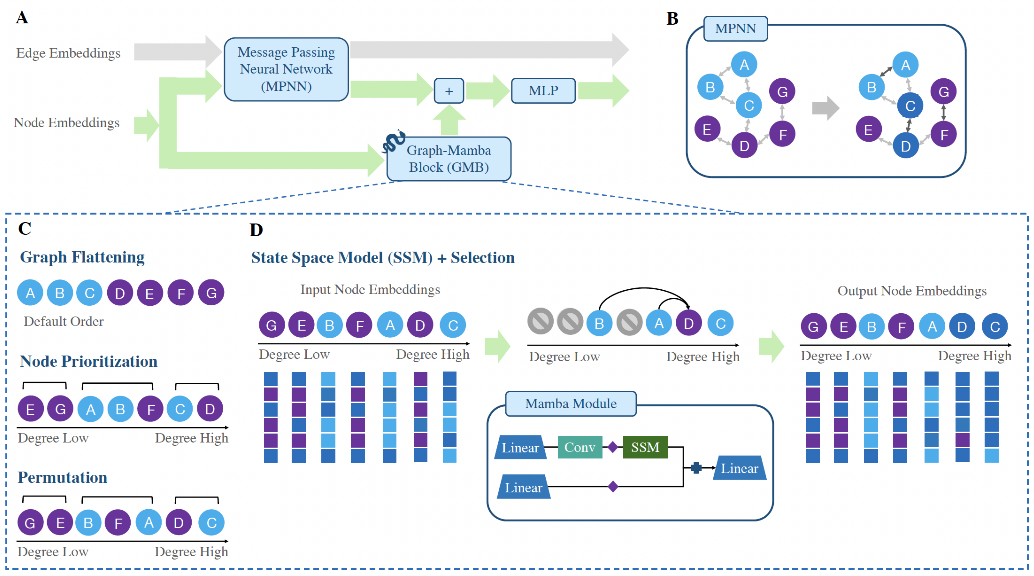


Figure 20: Architecture of Graph-Mamba (Image Source: [16])

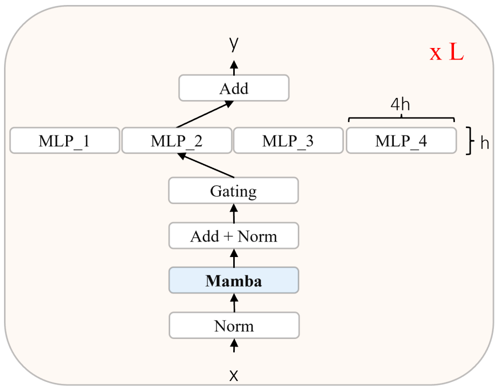


Figure 21: Architecture of Mamba-MoE (Image Source: [1])

platform provides an instruction fine-tuned version of Mamba model named **Mamba-Chat**. It showcases the processing of longer sequences more efficiently without computational complexity that scales quadratically with the input length. The Mamba-2.8B model has demonstrated impressive performance across various tasks. And Mamba-Chat's versatility is highlighted through its fine-tuning for specific applications, such as cybersecurity, showcasing its adaptability and potential in specialized knowledge domains.

4 Future Work

Transformer based foundation models (LLMs) have a rich ecosystem of AI and MLSys-centric research such as PeFT, Adaptation, Prompting, In-Context Learning, RLHF, Quantization, Distillation, Instruction Tuning and so on. The future research would focus more on scaling Mamba models to an extent of the Transformer models and studying the extended research work as highlighted above for SSM models.

As shown in current work, recent papers have showcased an interest in taking tbe Mamba models towards multi-modal systems such as, Mamba-ND and Graph-Mamba. Moreover, MoE (ensemble equivalent) models of Mamba SSMs have also gained some traction.

References

[1] Anthony, Q., Tokpanov, Y., Glorioso, P., and Millidge, B. Blackmamba: Mixture of experts for state-space models, 2024.

[2] Fu, D. Y., Dao, T., Saab, K. K., Thomas, A. W., Rudra, A., and Ré, C. Hungry hungry hippos: Towards language modeling with state space models, 2023.

[3] Gu, A., and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces, 2023.

[4] Gu, A., Dao, T., Ermon, S., Rudra, A., and Re, C. Hippo: Recurrent memory with optimal polynomial projections, 2020.

[5] Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces, 2022.

[6] Gu, A., Johnson, I., Goel, K., Saab, K., Dao, T., Rudra, A., and Ré, C. Combining recurrent, convolutional, and continuous-time models with linear state-space layers, 2021.

[7] Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention, 2020.

[8] Li, S., Singh, H., and Grover, A. Mamba-nd: Selective state space modeling for multi-dimensional data, 2024.

[9] Patel, C. Performance-efficient mamba-chat from nvidia ai foundation models. `https://developer.nvidia.com/blog/performance-efficient-mamba-chat-from-nvidia-ai-foundation-models/`. [Feb 12, 2024].

[10] Poli, M., Massaroli, S., Nguyen, E., Fu, D. Y., Dao, T., Baccus, S., Bengio, Y., Ermon, S., and Ré, C. Hyena hierarchy: Towards larger convolutional language models, 2023.

[11] Smith, J. T. H., Warrington, A., and Linderman, S. W. Simplified state space layers for sequence modeling, 2023.

[12] Tay, Y., Dehghani, M., Abnar, S., Shen, Y., Bahri, D., Pham, P., Rao, J., Yang, L., Ruder, S., and Metzler, D. Long range arena: A benchmark for efficient transformers, 2020.

[13] Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. Efficient transformers: A survey, 2022.

[14] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need, 2023.

[15] Voelker, A., Kajić, I., and Eliasmith, C. Legendre memory units: Continuous-time representation in recurrent neural networks. In *Advances in Neural Information Processing Systems* (2019), H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, Curran Associates, Inc.

[16] Wang, C., Tsepa, O., Ma, J., and Wang, B. Graph-mamba: Towards long-range graph sequence modeling with selective state spaces, 2024.