Master's Thesis

# State Space Models: An Efficient Alternative to Attention

Sampad Kumar Kar

Supervised by Prof. Pranabendu Misra

May 24, 2024

# Abstract

The Transformer architecture, with its core self-attention mechanism, has revolutionised deep learning, achieving state-of-the-art results across various tasks. However, the quadratic scaling of attention with sequence length presents a significant bottleneck for handling long sequences. This has driven the exploration of alternative sequence modelling paradigms, with state space models (SSMs) emerging as a promising contender. This survey delves into the recent advancements in SSMs, focusing on their theoretical foundations, key architectures, and empirical performance. We discuss prominent SSM variants, including S4, Mamba, Linear Recurrent Units (LRUs) and Griffin, highlighting their unique strengths in addressing challenges like computational efficiency and capturing long-range dependencies. We also present experimentals on toy datasets, demonstrating their performance in text, image, and audio generation using tiny LLMs.

# Contents

# Introduction

Recurrent Neural Networks (RNNs) have historically been the dominant paradigm for sequence modelling, achieving SoT results in tasks such as Machine Translation, Text Generation, and Speech Recognition. RNNs excel at capturing sequential information by iteratively updating a hidden state based on the current input and the previous state. However, RNNs are inherently limited in their ability to handle long-range dependencies (LRDs), as information from the distant past can be lost due to the vanishing gradient problem. This restriction led to the development of Long Short-Term Memory (LSTM) networks, which introduced gating mechanisms to selectively remember or forget information, mitigating some of the drawbacks of traditional RNNs.

Despite these advancements, RNNs and LSTMs still face challenges in capturing LRDs effectively and efficiently. Training these models is often slow due to their sequential nature, and they can struggle to generalise when the length of the input sequence varies.

The emergence of Transformers [20] as an alternative sequence modelling paradigm has significantly shifted the landscape. Transformers, with their core self-attention mechanism, allow each token to interact with every other token in the input sequence, enabling them to capture LRDs effectively. This has led to a surge in performance across various tasks, making Transformers the dominant paradigm in NLP and other sequence-to-sequence tasks.

However, Transformers are not without their shortcomings. The quadratic complexity of their self-attention mechanism, scaling as $\mathcal{O}(n^2)$ with sequence length $n$, poses a significant computational bottleneck, especially when dealing with long sequences in domains like genomics and high-resolution image analysis. Moreover, Transformers also struggle in the Long Range Arena (LRA) benchmark, a suite of challenging tasks designed to test models' ability to reason over very long sequences. This suggests that Transformers, despite their overall strength, might not be the ideal solution for all sequence modelling tasks, particularly those demanding efficient processing of long sequences.

State Space Models (SSMs) have emerged as a compelling alternative to Transformers, offering key efficiency advantages. SSMs, inspired by classical models from control theory and signal processing, compress input data into a fixed-size latent state during sequence generation. This static memory allocation enables SSMs to process long sequences efficiently without the quadratic complexity of attention. Furthermore, SSMs can be computationally advantageous during inference, exhibiting faster processing compared to Transformers.

This survey delves into the recent advancements in SSMs, focusing on their theoretical foundations, key architectures, and empirical performance. We ex-

amine prominent SSM variants, including S4 [7], Mamba [6], Linear Recurrent Units (LRUs) [13], and Griffin [2], highlighting their unique strengths in addressing challenges like computational efficiency and capturing LRDs. We also present experimental results on toy datasets, demonstrating their performance in text, image, and audio generation using tiny LLMs.

We organised the thesis into the chapters as follows:

- **Chapter 1** outlines the framework for deep sequence modelling, outlining the key challenges in this domain, including the LRA benchmark.
- **Chapter 2** provides a background on Transformers and the Attention mechanism and its limitations.
- **Chapter 3** introduces the SSM framework, discussing its conceptual foundation and fundamental principles. Then it delves into the S4 architecture, a prominent SSM variant, exploring its unique characteristics and how it overcomes the computational limitations of previous SSM models, including the core mechanics of S4.
- **Chapter 4** focuses on Mamba, a more recent SSM architecture, emphasising its selective state space and its ability to efficiently process long sequences.
- **Chapter 5** introduces LRUs and their theoretical universality. We discuss how LRUs can be designed to achieve comparable performance to SSMs with minimal architectural changes.
- **Chapter 6** examines Hawk and Griffin architecture, which combines the strengths of LRUs and local attention to achieve even higher performance and efficiency.
- **Chapter 7** presents some experimentals, showcasing the capabilities of SSMs on some toy datasets.

# Chapter 1

# Introduction to Deep Sequence Modelling

## 1.1 Deep Sequence Models

Deep learning methods have made significant advances in machine learning, achieving widespread success across scientific and industrial applications. A core class of models are sequence models, which are parameterised mappings operating on arbitrary sequences of inputs. These can be applied to a wide variety of complex sequential tasks such as:

- **Language Modelling**: The input is a sequence of tokens representing words, and the output is a probability distribution over the next word in the sequence.
- **Machine Translation**: The input is a sequence of tokens representing a sentence in one language, and the output is a sequence of tokens representing the translation in another language.
- **Speech Recognition**: The input is a sequence of audio samples representing speech, and the output is a sequence of tokens representing the transcribed text.
- **Image Captioning**: The input is a sequence of pixels representing an image, and the output is a sequence of tokens representing a textual description of the image.
- **Time Series Forecasting**: The input is a sequence of historical data points, and the output is a prediction of future data points.

**Definition 1.** *A **Sequence Model** is a parameterised map on sequences denoted as $y = f_\theta(x)$ where:*

- ***Inputs** $(x)$: A sequence of length $L$ of feature vectors in $\mathbb{R}^D$, i.e., $x \in \mathbb{R}^{L \times D}$. Here, $L$ represents the sequence length and $D$ represents the number of features or channels. Each element $x_k \in \mathbb{R}^D$ corresponds to the feature vector at position $k$ in the sequence.*

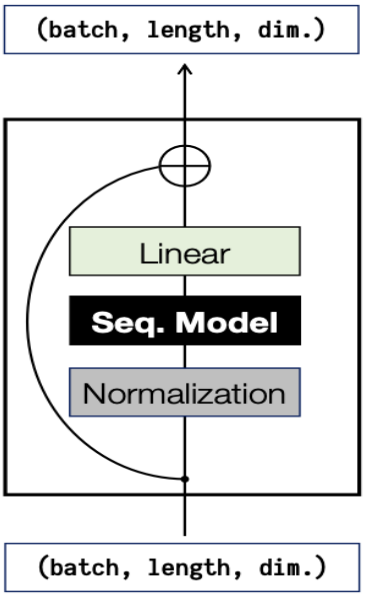



Figure 1.1: A deep sequence model is a neural network architecture built around a core sequence transformation such as convolutions, attention, or recurrence, and comprising additional components such as normalisation layers, linear layers, and residual connections. This boxed architecture block is usually composed repeatedly into a deep neural network.

- ***Outputs*** *(y): A sequence of length $L$ of feature vectors, i.e., $y \in \mathbb{R}^{L \times D'}$. The output dimension $D'$. Each element $y_k \in \mathbb{R}^{D'}$ corresponds to the output feature vector at position $k$.*
- ***Parameters*** *($\theta$): A collection of parameters learned through gradient descent.*

We can define different deep sequence models by replacing the core sequence model layer (which maps $\mathbb{R}^{B \times L \times D}$ to $\mathbb{R}^{B \times L \times D}$) [1] with various sequence modelling mechanisms. Here, $B$ represents the batch size. This framework allows us to explore different architectures and their strengths.

- **Attention** [20]: This is the core mechanism of Transformers, which allows for global interactions between all elements in a sequence. The complexity of attention scales quadratically with sequence length, making it inefficient for long sequences.
- **S4** [7]: The S4 model utilises a structured state space model (SSM) that leverages linear recurrences for efficient computations.
- **Mamba** [6]: The Mamba model also utilises SSMs but incorporates a selection mechanism. This mechanism enables the model to selectively propagate or forget information based on the current input token, resulting in more effective modelling.
- **Linear Recurrent Units (LRUs)** [13]: LRUs achieve comparable performance to SSMs by carefully designing deep RNNs using linear recurrences, specific parameterisations, and normalisation.
- **Hawk and Griffin** [2]: These architectures combine the strengths of LRUs and local attention to achieve even higher performance and efficiency.

[1] We are interested in cases where $D = D'$

Deep sequence models also face several challenges:

- **General-Purpose Capabilities**: Developing sequence models that can perform effectively across various domains and capabilities requires addressing the specific strengths and weaknesses of different model families.
- **Computational Efficiency**: Ensuring efficient computation of the sequence-to-sequence mapping is crucial, especially for long sequences and online/autoregressive settings.
- **Long-Range Dependencies**: Modelling complex interactions within long sequences is a key challenge. Existing approaches often struggle with capturing LRDs effectively and efficiently.

In order to test these models over LRDs, we introduce the **Long Range Arena** (LRA) Benchmark [17].

## 1.2 Long Range Arena

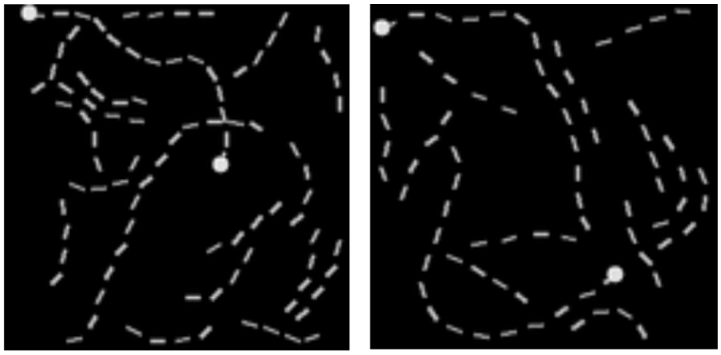
The LRA Benchmark, introduced by Tay et al. [17] is a suite of challenging tasks designed to test models' ability to reason over very long sequences. The benchmark consists of a diverse set of tasks, including:

- **Long ListOps**: This task evaluates the model's ability to perform nested mathematical operations on a sequence of up to 2K tokens. It tests the model's hierarchical reasoning capabilities over long contexts.
- **Byte-level Text Classification**: This task involves classifying text into categories at the byte/character level, challenging models to compose and aggregate information over longer sequences. The IMDb reviews dataset is used for this task, with a fixed maximum length of 4K tokens.
- **Byte-level Document Retrieval**: This task tests the model's ability to encode and store compressed representations of long sequences suitable for similarity-based matching. The ACL Anthology Network dataset is used, requiring the model to determine whether two papers have a citation link.
- **Image Classification on Sequences of Pixels**: The input is a sequence of pixels representing a flattened image, forcing the model to learn 2D spatial relations between pixels while presented with a 1D sequence. The CIFAR-10 dataset is used for this task.

Apart from these tasks, the Pathfinder and PathX tasks are particularly interesting due to their emphasis on long-range spatial dependency.

### 1.2.1 Pathfinder and PathX

Pathfinder (and its more challenging variant, PathX) was designed to probe the ability of models to learn long-range spatial dependencies. It presents a synthetic visual task where a model needs to decide whether two points (represented as circles) are connected by a path consisting of dashes. The task is designed to be



(a) A positive example. (b) A negative example.

Figure 1.2: Samples of Pathfinder task.

difficult due to the presence of distractor paths and variations in the path shape, requiring the model to reason over the entire image to identify the target contour and trace it from one end to the other.

- **Pathfinder**: The input images are $32 \times 32$ pixels, resulting in a sequence length of 1024.
- **PathX**: This variant increases the difficulty significantly by scaling up the image resolution to $128 \times 128$, resulting in a sequence length of 16384.

At the time LRA was proposed, no model could successfully solve the PathX task, with most models achieving no better than random guessing. This highlights the computational and conceptual challenges posed by models in reasoning over such long sequences. **S4 [7] was the first model to successfully crack the PathX task**, achieving remarkable performance on the LRA benchmark. This breakthrough underscored the potential of SSMs as a powerful alternative to Transformers [20], particularly in handling long sequences and challenging LRD tasks.

# Chapter 2

# Transformers and the Attention Mechanism

## 2.1 Transformers and Attention

The Transformer architecture, introduced by Vaswani et al. [20] is based on an encoder-decoder structure, typically using a stacked architecture of encoder and decoder blocks.

### 2.1.1 Encoder Block

An encoder block generally comprises the following components, sequentially applied to the input sequence:

- **Multi-Head Self-Attention Layer**: This layer computes global interactions between all tokens within a sequence, capturing long-range dependencies.
- **Feedforward Network**: This layer performs position-wise transformations on the output of the attention layer, enhancing the representation of each token.
- **Residual Connections**: These connections skip layers, preventing vanishing gradients and allowing information flow from earlier layers to later layers.
- **Layer Normalisation**: This normalisation technique helps stabilise training and improve generalisation by scaling the output of each layer.

### 2.1.2 Decoder Block

Similar to the encoder block, the decoder block typically includes:

- **Masked Multi-Head Self-Attention Layer**: This layer is masked to ensure that the decoder attends only to previously generated tokens, maintaining the autoregressive property.

- **Encoder-Decoder Attention Layer**: This layer allows the decoder to attend to the output of the encoder, incorporating information from the input sequence.
- **Feedforward Network**: This layer performs position-wise transformations on the output of the attention layers, similar to the encoder block.
- **Residual Connections**: These connections skip layers, preventing vanishing gradients and allowing information flow from earlier layers to later layers.
- **Layer Normalisation**: This normalisation technique helps stabilise training and improve generalisation by scaling the output of each layer.

The output of the final decoder layer is then typically passed through a linear projection and a softmax layer to generate a probability distribution over the vocabulary, which is used for tasks such as text generation.

## 2.2 Attention Mechanism

The attention mechanism [20] is the core of the Transformer architecture. It allows the model to dynamically focus on specific parts of the input sequence, effectively weighting the importance of different tokens when generating the output. The most common form is self-attention, where the model attends to itself.

### 2.2.1 Self-Attention

Let's define self-attention mathematically:

- **Input sequence**: $x = (x_1, x_2, ..., x_n) \in \mathbb{R}^{n \times d}$ represents a sequence of $n$ tokens, where each token $x_i \in \mathbb{R}^d$ is a $d$-dimensional vector.
- **Query, Key, and Value Matrices**: The input sequence is projected into three matrices:
  - **Query Matrix**: $Q \in \mathbb{R}^{n \times d_k}$ is a projection of the input sequence, where $d_k$ is the key dimension.
  - **Key Matrix**: $K \in \mathbb{R}^{n \times d_k}$ is another projection of the input sequence, where $d_k$ is the key dimension.
  - **Value Matrix**: $V \in \mathbb{R}^{n \times d_v}$ is a projection of the input sequence, where $d_v$ is the value dimension.
- **Attention Weights**: The attention weights are computed by taking the dot product of the query matrix with the key matrix, followed by a softmax normalization:
  - **Similarity Matrix**: $S = QK^T \in \mathbb{R}^{n \times n}$
  - **Attention Weights**: $A = softmax(S) \in \mathbb{R}^{n \times n}$

- **Output**: Finally, the attention output is computed by multiplying the attention weights with the value matrix:
  - **Attention Output**: $O = AV \in \mathbb{R}^{n \times d_v}$.

This process allows the model to dynamically attend to specific tokens in the input sequence, effectively weighing the contribution of each token to the output.

### 2.2.2 Scaled Dot-Product Attention

A common implementation of the attention mechanism is Scaled Dot-Product Attention [20]. It introduces a scaling factor to the dot product, dividing by $\sqrt{d_k}$, to prevent the dot products from becoming too large and causing numerical instability in the softmax function. This results in the following equation:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

### 2.2.3 Multi-Head Attention

To further enhance the expressivity of the attention mechanism, the Multi-Head Attention [20] architecture is used. It performs multiple attention calculations in parallel, with each head focusing on different aspects of the input. The outputs of each head are then concatenated and projected to form the final attention output. This is commonly defined with $h$ heads as follows:

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O$$

where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$, with the learnable projection matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}$, and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$.

## 2.3 Limitations of Attention

While Transformers with their attention mechanism have achieved impressive performance they still have a few limitations:

- **Quadratic Complexity**: The self-attention in Transformers requires computing a similarity matrix between all token pairs in the input sequence, resulting in a complexity of $\mathcal{O}(n^2)$. This means the computational cost and memory requirements grow quadratically with the sequence length $n$. This makes it computationally expensive to process long sequences, especially in domains like genomics and high-resolution image analysis.
- **Inference Speed**: Inference speed of Transformers still scales linearly with the sequence length. This is because the KV cache [16], storing key and value vectors for each token, grows linearly with the sequence length. This can make inference slow and impractical for real-time applications or those requiring the processing of very long sequences.

- **Long Range Arena Benchmark**: Until very recently, the computational limitations of Transformers prevented them from successfully tackling the challenging **PathX** task in the LRA benchmark [17]. This task, with a sequence length of 16384, necessitates capturing long-range spatial dependencies, highlighting the difficulty faced by Transformers in reasoning over very long sequences. Only with the advancement in GPUs and compute power did Transformers start to achieve decent results on this benchmark.

These limitations have motivated researchers to explore alternative sequence modelling paradigms, with SSMs emerging as a promising contender that can address these challenges.

# Chapter 3

# SSMs: Foundations and the S4 Architecture

The Transformer architecture, with its core self-attention mechanism, has revolutionized deep learning, achieving state-of-the-art results across a wide range of tasks. However, as discussed in **Section 2.3**, the quadratic scaling of attention with sequence length poses a significant bottleneck for handling long sequences. This has driven the exploration of alternative sequence modeling paradigms, with SSMs emerging as a promising contender.

SSMs, inspired by classical models from control theory and signal processing, offer a continuous-time framework for capturing the dynamics of sequences. The core idea is to represent sequential data through a latent state that evolves over time, allowing for efficient representation and processing of long sequences.

The next few chapters dive into the recent advancements in SSMs, focusing on their theoretical foundations and their practical implementation. We start by exploring the S4 architecture [7], a prominent SSM variant, and explain how it overcomes the computational limitations of previous SSM models.

## 3.1 State Space Models: A Formal Introduction

The State Space Model (SSM) is a versatile mathematical framework for representing and analysing dynamic systems. It offers a way to capture complex system behaviour using a set of first-order differential or difference equations. These equations describe the evolution of the system's internal state, which is represented by a vector of state variables.

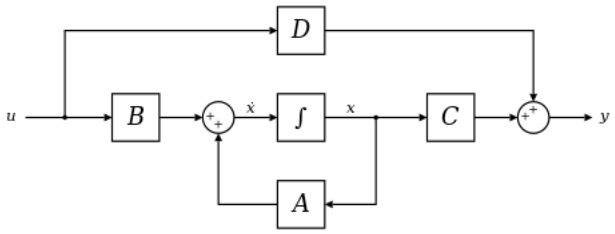



Figure 3.1: Block diagram representation of Linear SSM

### 3.1.1 Continuous-Time SSMs

A continuous-time SSM is defined by the following system of differential equations:

$$\dot{\mathbf{x}}(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t)$$

where:

- $\mathbf{x}(t) \in \mathbb{R}^n$ represents the system's state vector at time $t$, encapsulating its internal condition.
- $\mathbf{u}(t) \in \mathbb{R}^p$ represents the input vector or control variables at time $t$.
- $\mathbf{y}(t) \in \mathbb{R}^k$ represents the output vector containing the measurable quantities of interest at time $t$.
- $\dot{\mathbf{x}}(t) = \frac{d}{dt}\mathbf{x}(t)$ is the time derivative of the state vector.
- $\mathbf{A}(t) \in \mathbb{R}^{n\times n}$ is the state matrix, defining the system's internal dynamics.
- $\mathbf{B}(t) \in \mathbb{R}^{n\times p}$ is the input matrix, mapping the input to the state.
- $\mathbf{C}(t) \in \mathbb{R}^{k\times n}$ is the output matrix, mapping the state to the output.
- $\mathbf{D}(t) \in \mathbb{R}^{k\times p}$ is the feedthrough matrix, directly mapping the input to the output, like a skip connection.

### 3.1.2 Linear Time-Invariant SSMs

In the linear time-invariant (LTI) case, the matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ remain constant over time, leading to a simpler representation:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

These LTI systems have a crucial connection to convolutions, which is key to their efficient computation. We'll explore this relationship in more detail in the next section, while discussing S4.

### 3.1.3 Discrete-Time SSMs

Discrete-time SSMs model the evolution of a system at discrete time steps, representing the state and input as sequences. We are particularly interested in this, because they operate on discrete sequences of data, such as audio samples, image pixels, or text tokens.

A discrete-time invariant SSM is defined as follows:

$$\mathbf{x}(t+1) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$$
$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t)$$

where $t \in \mathbb{N}$ represents the discrete time steps, and the other terms are identical to the continuous-time SSM.

To connect the continuous-time SSM to its discrete-time counterpart, we need to use a discretization procedure. Discretization involves transforming the continuous-time system parameters ($\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$) into their discrete-time equivalents. This transformation is usually achieved using numerical integration methods. We discuss three commonly used discretization techniques in the upcoming sections.

### 3.1.4 Euler Method

The Euler method is a first-order numerical integration technique that approximates the derivative of a function using a forward difference. For the continuous-time SSM, the Euler method yields the following discrete-time update (for $\Delta \simeq 0$):

$$\begin{aligned}\mathbf{x}(t+\Delta) &\simeq \mathbf{x}(t) + \Delta\dot{\mathbf{x}}(t) \\ &= \mathbf{x}(t) + \Delta(\mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)) \\ &= (\mathbf{I} + \Delta\mathbf{A})\mathbf{x}(t) + \Delta\mathbf{B}\mathbf{u}(t)\end{aligned}$$

where $\Delta$ is the time step and $\mathbf{I}$ is the identity matrix. The output equation remains unchanged. Thus, the Euler method produces the discretized matrices as:

$$\begin{aligned}\overline{\mathbf{A}} &= (\mathbf{I} + \Delta\mathbf{A}) \\ \overline{\mathbf{B}} &= \Delta\mathbf{B}.\end{aligned}$$

### 3.1.5 Bilinear Transform

The bilinear transform offers improved accuracy over the Euler method. Applying the bilinear transform to the continuous-time SSM results in the following discrete-time update (for $\Delta \simeq 0$):

$$\mathbf{x}(t+\Delta) \simeq \left(\mathbf{I} - \frac{\Delta}{2}\mathbf{A}\right)^{-1}\left(\mathbf{I} + \frac{\Delta}{2}\mathbf{A}\right)\mathbf{x}(t) + \left(\mathbf{I} - \frac{\Delta}{2}\mathbf{A}\right)^{-1}\Delta\mathbf{B}\mathbf{u}(t).$$

This gives the discretized matrices as:

$$\begin{aligned}\overline{\mathbf{A}} &= \left(\mathbf{I} - \frac{\Delta}{2}\mathbf{A}\right)^{-1}\left(\mathbf{I} + \frac{\Delta}{2}\mathbf{A}\right) \\ \overline{\mathbf{B}} &= \left(\mathbf{I} - \frac{\Delta}{2}\mathbf{A}\right)^{-1}\Delta\mathbf{B}.\end{aligned}$$

### 3.1.6 Zero-Order Hold (ZoH)

The ZoH method approximates the input signal as constant between time steps. This approach is simpler to implement and can be more efficient, but it might introduce some approximation errors. For the continuous-time SSM, the ZoH method yields the following discrete-time update (for $\Delta \simeq 0$):

$$\mathbf{x}(t+\Delta) \simeq e^{\Delta\mathbf{A}}\mathbf{x}(t) + (\Delta\mathbf{A})^{-1}(e^{\Delta\mathbf{A}} - \mathbf{I})(\Delta\mathbf{B})\mathbf{u}(t)$$

This gives the discretized matrices as:

$$\begin{aligned}\overline{\mathbf{A}} &= e^{\Delta\mathbf{A}} \\ \overline{\mathbf{B}} &= (\Delta\mathbf{A})^{-1}(e^{\Delta\mathbf{A}} - \mathbf{I})(\Delta\mathbf{B}).\end{aligned}$$

In the next section, we will focus on the S4 model, which is an SSM variant that utilizes the bilinear transform for discretization.

## 3.2 S4

The S4 model, introduced by Gu et al. [7], stands out as a significant advancement in the field of SSMs. It addresses the computational limitations of previous SSM models, leading to a more efficient and effective approach to long-range dependency modelling.

As discussed in **Section 3.2**, LTI SSMs are particularly interesting because they have a close connection to convolutions, which allows for efficient parallel training. The **S4 is a type of LTI SSM**. This means that its state matrix $\mathbf{A}$, input matrix $\mathbf{B}$, and output matrix $\mathbf{C}$ are all constant across time steps.

To translate the continuous-time S4 model into a discrete-time form suitable for computation, the **Bilinear Transform** (discussed in **Section 3.1.5**) is employed. This choice is particularly beneficial for S4, as it allows for a more stable and efficient representation of the model.

### 3.2.1 Training S4: Convolutional Representation

LTI models (hence S4) can be expressed in terms of a convolution operation. This is a key advantage as it allows for parallelized training and efficient computations. Let's derive the convolutional representation for the LTI models.

Starting with the discrete-time SSM:[1]

$$\begin{aligned}\mathbf{x}(t+1) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)\\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t)\end{aligned}$$

Assuming an initial state of $\mathbf{x}(-1) = 0$, we can unroll the recurrence to get:

$$\begin{aligned}\mathbf{x}(0) &= \mathbf{B}\mathbf{u}(0)\\ \mathbf{x}(1) &= \mathbf{A}\mathbf{B}\mathbf{u}(0) + \mathbf{B}\mathbf{u}(1)\\ \mathbf{x}(2) &= \mathbf{A}^2\mathbf{B}\mathbf{u}(0) + \mathbf{A}\mathbf{B}\mathbf{u}(1) + \mathbf{B}\mathbf{u}(2)\\ &\vdots\end{aligned}$$

and similarly, the output:

$$\begin{aligned}\mathbf{y}(0) &= \mathbf{C}\mathbf{B}\mathbf{u}(0)\\ \mathbf{y}(1) &= \mathbf{C}\mathbf{A}\mathbf{B}\mathbf{u}(0) + \mathbf{C}\mathbf{B}\mathbf{u}(1)\\ \mathbf{y}(2) &= \mathbf{C}\mathbf{A}^2\mathbf{B}\mathbf{u}(0) + \mathbf{C}\mathbf{A}\mathbf{B}\mathbf{u}(1) + \mathbf{C}\mathbf{B}\mathbf{u}(2)\\ &\vdots\end{aligned}$$

This can be expressed as a convolution with kernel $\mathbf{K}$:

$$\mathbf{y} = \mathbf{u} * \mathbf{K}$$

where

$$\mathbf{K} = (\mathbf{C}\mathbf{B}, \mathbf{C}\mathbf{A}\mathbf{B}, \mathbf{C}\mathbf{A}^2\mathbf{B}, \dots)$$

This convolutional form is crucial because it allows for efficient parallelized computation by pre-computing the kernel $\mathbf{K}$, and allows us to avoid the recurrence of LTIs.

[1] Here we are ignoring $\mathbf{D}$ because it is just a skip connection and does not affect the complexity of the calculation meaningfully.

### 3.2.2 Challenges with Naive Implementation

The naive implementation of the convolution kernel involves repeatedly multiplying the matrix $\mathbf{A}$ by the vector $\mathbf{B}$. This requires $\mathcal{O}(N^2L)$ operations and $\mathcal{O}(NL)$ space for a state dimension $N$ and sequence length $L$. This complexity quickly becomes prohibitive as the sequence length grows, especially for larger SSMs (where $N$ might be 100 or more).

### 3.2.3 Adding Further Structure to A

To address the computational challenges of computing $\mathbf{K}$, Gu et al. [7] propose adding structure to the state matrix $\mathbf{A}$. This structure makes it possible to efficiently compute both the convolutional representation ($\mathbf{K}$) and the recurrent update rule for the state vector. The key idea behind S4 is to use a **Normal Plus Low-Rank** (NPLR) representation for $\mathbf{A}$, which we discuss next.

#### Using HIPPO Matrix

Gu et al. [8, 7] proposed a specific structure for $\mathbf{A}$, known as the **HIPPO matrix**, which has a remarkable connection to orthogonal polynomial projections and time-invariant SSMs. The HIPPO matrix is defined as:

$$\mathbf{A}_{nk} = -\begin{cases} (2n+1)^{\frac{1}{2}}(2k+1)^{\frac{1}{2}} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases}$$

This matrix has the NPLR property that allows for efficient computation of the SSM kernel, even for long sequences. Its NPLR form is as follows:

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^* - \mathbf{P}\mathbf{Q}^T,$$

where:

- $\mathbf{V} \in \mathbb{C}^{\mathbf{N}\times\mathbf{N}}$ is a unitary matrix,
- $\mathbf{\Lambda}$ is a diagonal matrix,
- $\mathbf{P}, \mathbf{Q} \in \mathbb{R}^{\mathbf{N}\times r}$, with $r = 1$ for HIPPO.

#### Complexity Improvements

As shown by the authors in the S4 paper [2], this NPLR parameterisation of the $\mathbf{A}$ matrix leads to significant improvements in computational complexity, both in recurrent and convolutional modes:

- **Recurrent Mode Complexity**: Computing one step of the recurrence with a NPLR matrix $\mathbf{A}$ requires only $\mathcal{O}(\mathbf{N})$ operations, making it computationally efficient.

[2]Refer to **Theorem 2** and **Theorem 3** of **Section 3.3** of the S4 Paper [7]

- **Convolution Mode Complexity**: The S4 kernel computation involves evaluating the generating function of $\mathbf{K}$ at specific points, which can be done efficiently in $\tilde{\mathcal{O}}(\mathbf{N}+\mathbf{L})$ operations (where $L$ is the sequence length) using a black-box Cauchy kernel computation. This complexity makes it optimal for efficient kernel pre-computation and helps in avoiding the recurrence of S4.

| | **Convolution** | **Recurrence** | **Attention** | **S4** |
|---|---|---|---|---|
| **Parameters** | $LH$ | $H^2$ | $H^2$ | $H^2$ |
| **Training** | $\tilde{L}H(B+H)$ | $BLH^2$ | $B(L^2H+LH^2)$ | $BH(\tilde{H}+\tilde{L})+B\tilde{L}H$ |
| **Space** | $BLH$ | $BLH$ | $B(L^2+HL)$ | $BLH$ |
| **Parallel** | Yes | No | Yes | Yes |
| **Inference** | $LH^2$ | $H^2$ | $L^2H+H^2L$ | $H^2$ |

Table 3.1: Complexity of various sequence models in terms of sequence length ($L$), batch size ($B$), and hidden dimension ($H$); tildes denote log factors. Metrics are computation for 1 sample and time-step. For simplicity, the state size of S4 $N = H$. Convolutions are efficient for training while recurrence is efficient for inference, while SSMs combine the strengths of both.

### 3.2.4 Architecture of Deep S4

The S4 model can be incorporated into deep neural network architectures, similar to other sequence models, to improve its expressivity and ability to learn complex features. Here are some key aspects of the deep S4 architecture:

- **Multi-Channel Support**: For inputs with multiple features (channels/dimensions), S4 uses separate, independently trained models for each channel.
- **Parameter Tying**: The state matrix $\mathbf{A}$, input matrix $\mathbf{B}$, and timescale parameter $\Delta$ can be tied across different channels to reduce the number of parameters.
- **Linear Mixing**: After each S4 layer, a linear feed-forward layer is applied to mix the outputs from different channels.
- **Nonlinear Activations**: A non-linear activation function, such as GELU, is commonly used between S4 layers.
- **Residual Connections**: Residual connections are employed to improve information flow and mitigate vanishing gradients in deep networks.
- **Normalization**: Layer Normalization or Batch Normalization is often used to stabilize training and enhance generalization.

### 3.2.5 Empirical Performance in LRA

The S4 model has achieved significant performance improvements on the LRA benchmark [17], particularly for tasks requiring long-range dependencies. It has demonstrated the **first successful solution** for the challenging Path-X task[3] in the LRA benchmark as shown in 3.2. This showcases the S4 model's effectiveness in addressing long-range dependency challenges.

| Model | ListOps | Text | Retrieval | Image | Pathfinder | Path-X | Avg |
|---|---|---|---|---|---|---|---|
| Transformer | 36.37 | 64.27 | 57.46 | 42.44 | 71.40 | ✗ | 53.66 |
| Reformer | 37.27 | 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 | 65.90 | 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 | 77.80 | ✗ | 54.42 |
| Nyströmformer | 37.15 | 65.52 | 79.56 | 41.58 | 70.94 | ✗ | 57.46 |
| Luna-256 | 37.25 | 64.57 | 79.29 | 47.38 | 77.72 | ✗ | 59.37 |
| **S4** | **59.60** | **86.82** | **90.90** | **88.65** | **94.20** | **96.35** | **86.09** |

Figure 3.2: Original Transformer (Top) variants vs S4 (Bottom) in LRA

### 3.2.6 Limitations of S4 and LTIs in Language Modelling

While S4 excels in various domains, it has limitations in language modelling. This is primarily due to its inability to perform content-based reasoning, which is crucial for language modelling tasks. Here are two key examples illustrating this limitation:

- **Selective Copying**: S4 struggles with tasks like Selective Copying, where it needs to selectively remember and copy relevant tokens while filtering out irrelevant ones.
- **Induction Heads**: S4 also faces difficulties with Induction Heads tasks, which require the model to recall specific tokens based on contextual cues.

To address these limitations in LTIs, more recent SSM Models like Mamba [6] were developed.

[3] more details in **Section 1.2.1**

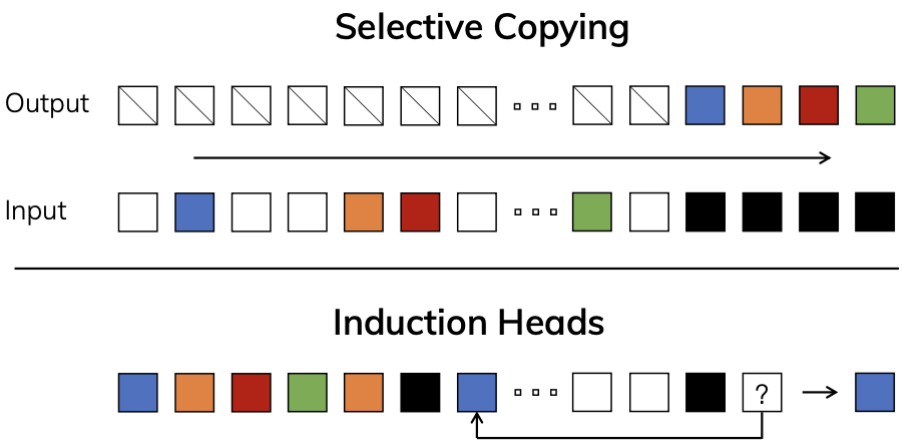



Figure 3.3: The standard version of the Copying task involves constant spacing between input and output elements and is easily solved by LTIs. (Top) The Selective Copying task has random spacing in between inputs and requires time-varying models that can selectively remember or ignore inputs depending on their content. (Bottom) The Induction Heads task is an example of associative recall that requires retrieving an answer based on context, a key ability for LLMs.

# Chapter 4

# Selective SSM and Mamba

While S4 [7] provides a powerful framework for capturing LRDs in sequence modelling, it has limitations in certain data modalities, especially those involving discrete and information-dense sequences like text. This is primarily due to S4's reliance on time-invariant (LTI) dynamics, where the model parameters are fixed across all time steps. This constraint hinders S4's ability to reason about the specific content of the sequence and make input-dependent decisions.

This chapter introduces Selective State Space Models (S6) [6], a class of SSMs that address this limitation by incorporating a selection mechanism. This mechanism allows the model to dynamically adjust its parameters based on the current input token, enabling it to selectively propagate or forget information along the sequence length dimension. We begin by discussing how to modify S4 to introduce this selection mechanism, emphasizing the key differences in its computation and parameterization. We then present Mamba [6], a simplified SSM architecture incorporating selective state spaces and a specialized block design that eliminates the need for attention or even MLP layers.

## 4.1 S6

The S6 model [6] is a modified version of S4 [7] that incorporates a selection mechanism to address the limitations of LTI SSMs, especially in handling discrete modalities like text. This section delves into the key aspects of S6.

### 4.1.1 Discretization Strategy: ZoH

While S4 utilizes the bilinear transform for discretization, S6 adopts the Zero-Order Hold (ZoH) method. This shift offers a more convenient implementation for introducing selectivity and enables better understanding of the selection mechanism.

Recall from **Section 3.1.6** that the ZoH method assumes a constant input signal between discrete time steps. This results in the following discretized update equation for S6:

$$\mathbf{x}(t+\Delta) \simeq e^{\Delta\mathbf{A}}\mathbf{x}(t) + (\Delta\mathbf{A})^{-1}(e^{\Delta\mathbf{A}} - \mathbf{I})(\Delta\mathbf{B})\mathbf{u}(t)$$

where:

- $\overline{\mathbf{A}} = e^{\Delta \mathbf{A}}$ is the discretized state matrix,
- $\overline{\mathbf{B}} = (\Delta \mathbf{A})^{-1}(e^{\Delta \mathbf{A}} - \mathbf{I})(\Delta \mathbf{B})$ is the discretized input matrix,
- $\Delta$ is the time step.

### 4.1.2 Breaking LTI: Adding Selectivity

The key to S6 lies in making the model's parameters time-dependent and input-aware. This is achieved by making the timescale parameter $\Delta$ and the input and output projection matrices $\mathbf{B}$ and $\mathbf{C}$ functions of the current input token.

Specifically, let $\mathbf{u}$ denote the input. The S6 update equation with selectivity is given as:

$$\begin{aligned} \mathbf{x}(t+1) &= \overline{\mathbf{A}}(t)\mathbf{x}(t) + \overline{\mathbf{B}}(t)\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t) \end{aligned}$$

where:

- $\Delta(t)$ is the timescale parameter, also a function of the current input token (with shape $(B, L, D)$).
- $\overline{\mathbf{A}}(t)$ is the discretized state matrix [1] (now time-dependent because of it's dependence on $\Delta(t)$).
- $\overline{\mathbf{B}}(t)$ is the discretized input matrix (now time-dependent because of it's dependence on time dependent $\mathbf{B}(t)$ and $\Delta(t)$).
- $\mathbf{C}(t)$ and $\mathbf{D}(t)$ are also time-dependent by initialization.

Specifically, in the S6 layer, $\mathbf{B}(t)$, $\mathbf{C}(t)$, $\mathbf{D}(t)$ and $\Delta(t)$ are linear projections of the current input token $\mathbf{u}(t)$ as follows:

- $\mathbf{B}(t) = Linear_N(\mathbf{u}(t))$,
- $\mathbf{C}(t) = Linear_N(\mathbf{u}(t))$,
- $\mathbf{D}(t) = Linear_N(\mathbf{u}(t))$
- $\Delta(t) = Broadcast_D(Linear_1(\mathbf{u}(t)))$.

Here, $Linear_d$ represents a parameterized (and learnable) linear projection to dimension $d$, and $Broadcast_D$ expands the output of a linear layer to dimension $D$ by replicating each element.

### 4.1.3 Adding Structure to A

Even though the state matrix $\mathbf{A}$ remains constant, the time-dependent nature of the discretization step introduces input-dependence to the discretized $\mathbf{A}$ matrix. This requires us to impose structure on $\mathbf{A}$ to ensure efficient computation. A recent work by Gu et al. [9] showed that diagonal state spaces are as effective as the structured state spaces used in S4 [7].

[1] $\mathbf{A}$ matrix is still constant with fixed initialization

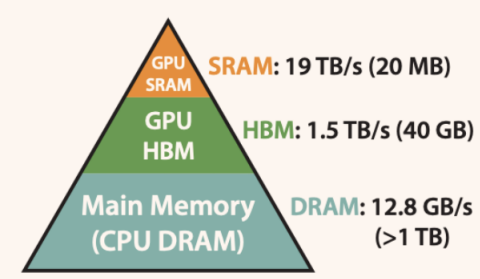



Figure 4.1: Memory Heirarchy with Bandwidth and Size

- **S4D-Lin-Complex and S4D-Real Initializations**: For S6, we typically use the S4D-Lin-Complex or S4D-Real initializations for the state matrix $\mathbf{A}$, derived from the HIPPO framework [8]. These initializations are well-suited for capturing long-range dependencies and offer a balance between expressivity and computational efficiency.
  - **S4D-Lin-Complex**: This initialization sets the diagonal elements of $\mathbf{A}$ as $A_k = -1/2 + ik$.
  - **S4D-Real**: This initialization sets the diagonal elements of $\mathbf{A}$ as $A_k = -(k+1)$. It is a real-valued initialization that avoids complex arithmetic and can achieve comparable performance. The selection of complex or real initializations depends on the specific task and its computational constraints.
- **Improving Recurrence with Associative Parallel Scan and Hardware Optimizations**: The selective nature of S6 prevents the use of efficient convolution-based computations enabled by LTIs. Therefore, the recurrent computation using a parallel scan [6] is adopted. However, directly applying the scan operation to the full state matrix can be inefficient due to its large size of scan input $(\overline{\mathbf{A}}, \overline{\mathbf{B}})$ of size $(B, L, D, N)$. To mitigate this, a hardware-aware algorithm is used that leverages the GPU's memory hierarchy.

  Concretely, instead of preparing the scan input in GPU High-Bandwidth Memory (HBM), we directly load the SSM parameters, i.e. $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ and $\Delta$ directly from slow HBM to fast SRAM, to perform the discretization and recurrence directly in SRAM and write the results back to HBM. This minimizes memory transfers between HBM and SRAM, thereby enhancing speed [2]. This is achieved via:
  - **Kernel Fusion**: The discretization, scan, and output projection operations are fused into a single kernel, reducing memory I/O operations and improving speed.
  - **Recomputation**: Instead of storing the intermediate states, which consume significant memory, they are recomputed during the backward pass. This minimizes memory overhead and accelerates training.

[2]This is similar to what happens in the FlashAttention [1] implementation of Transformers

## 4.2 Mamba

The Mamba model [6] builds upon the S6 architecture (which incorporates selection and scan mechanisms into S4) and takes a step further by simplifying the overall architecture and introducing a novel, hardware-aware implementation. This section delves into the core aspects of Mamba.

### 4.2.1 Architecture

Mamba simplifies the architecture of traditional SSM models, eliminating the need for attention or even vanilla MLP blocks. It achieves this by combining the SSM block with a gated MLP block into a single, homogeneous block design. This design effectively integrates the strengths of both SSMs and MLPs, resulting in a more efficient and streamlined architecture.

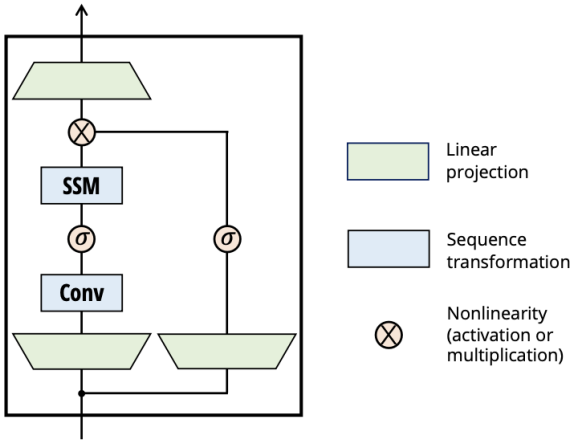



Figure 4.2: This represents a single Mamba Block, which can be stacked to form a deep Mamba model.

The Mamba Block, as shown in 4.2 consists of two branches:

- **SSM Branch**: The main branch processes the input sequence using a selective SSM layer (S6).
- **MLP Branch**: A gated MLPs block is applied in parallel to the SSM branch. This block uses a gated mechanism (SwiGLU) to enhance its expressivity and performance in long sequence modeling.

The outputs of both branches are then combined through element-wise multiplication, followed by a final linear projection. This design allows for efficient processing of long sequences while leveraging the computational advantages of both SSMs and MLPs.

### 4.2.2 Empirical Results

Mamba has achieved impressive results across diverse tasks and modalities, showcasing its potential as a general sequence modeling backbone. We highlight two key areas:

### Scaling Laws

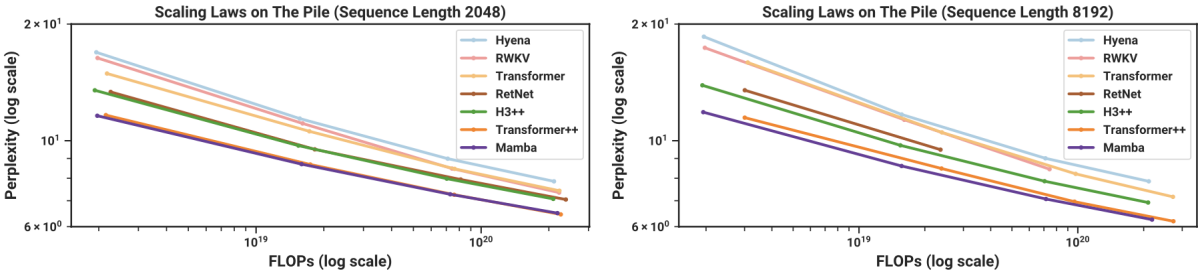



Figure 4.3: Scaling Laws of models with sizes from $125M - 1.3B$ parameters, trained on Pile Dataset [4].

As evident from 4.3, Mamba seems to scale better than all other attention-free models (like Hyena and RWKV) and is the first to match the performance of a very strong Transformer recipe[3] (which they call Transformer++).

### Efficiency Benchmarks

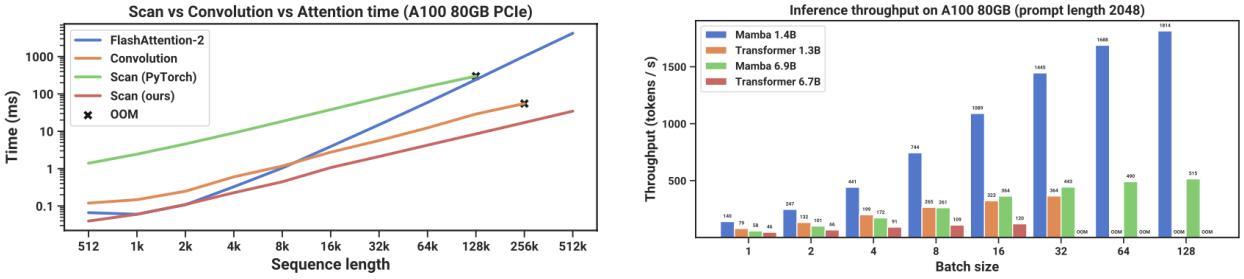



Figure 4.4: (Left) Training Times (Right) Inference Times of Mamba vs Transformers

From 4.4, we can see that Mamba is significantly faster than Transformers, especially during inference. The hardware-aware algorithm for the parallel scan[4] operation makes it faster than the standard scan implementation in PyTorch, promising a speedup of almost 20-40x. In terms of inference throughput, Mamba achieves remarkably higher throughput than Transformers of the same size, promising 4-5x higher inference throughput with similar memory footprint.

These empirical results strongly suggest that Mamba offers a compelling alternative to Transformers, particularly in domains demanding efficient long-sequence modeling.

[3]This like LLaMA [19] using RoPE, SwiGLU MLP, RMSNorm (instead of LayerNorm), no linear bias, and higher Learning Rate

[4]The S6 Recurrence in Mamba

# Chapter 5

# Linear Recurrent Units: Simplicity and Universality

The impressive performance of SSMs on LRD tasks sparked a natural question: **"Could this level of performance be achieved with a simpler model?"** Researchers turned their attention to RNNs with linear recurrences, investigating if careful design and parameterization, guided by principles learnt from S4's success, could reach comparable results without the complex mathematical machineries of SSMs.

This chapter delves into this exploration, starting with a discussion on the theoretical universality of RNNs when combined with non-linear projections. We then introduce Linear Recurrent Units (LRUs) [13], showcasing how a systematic series of modifications to vanilla RNNs can lead to comparable performance to S4 on the LRA benchmark [17].

## 5.1 Universality of Linear RNN followed by Non-Linear Projections

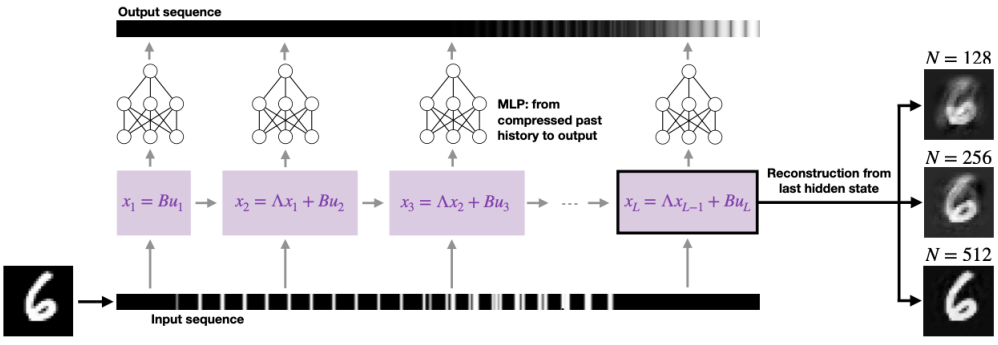



Figure 5.1: Example: Linear RNN + position-wise MLP for flattened MNIST digits reconstruction. The role of linear RNN is to compress and store the input sequence into the hidden states, from which we can recover past tokens using a MLP. As hidden size $N$ increases, the reconstruction becomes more and more faithful.

Orvieto et al. [14] proved that even basic architectures, combining linear RNNs and position-wise MLPs, are theoretically universal function approximators for

sequences. Their key insight is that these two components have distinct roles: the linear RNN acts as a compressor, encoding the entire input history into a fixed-size hidden state, while the MLP acts as a non-linear processor, extracting complex patterns from this compressed representation.

This universality hinges on two conditions:

- **Proper Initialization**: The RNN's state matrix must be initialized such that its eigenvalues lie close to the unit circle in the complex plane. This ensures the RNN can effectively capture long-range dependencies.
- **Sufficient Width**: Both the RNN and MLP need to be wide enough to accommodate the complexity of the target function.

This result highlights:

- **Simplicity and Efficiency**: Linear recurrences, combined with MLPs, are sufficient for universality, which can lead to computationally efficient models due to the parallelizability of linear RNNs[1].
- **Power of Complex Numbers**: Using complex values in the recurrence can improve the stability and effectiveness of the RNN's compression by allowing more control over the eigen-values.

## 5.2 LRU

The Linear Recurrent Unit (LRU), introduced by Orvieto et al. [13], is a carefully designed RNN architecture that leverages the theoretical universality of linear recurrences combined with non-linear projections to achieve strong performance. The LRU achieves this by systematically incorporating a series of modifications to a vanilla RNN, each designed to enhance its stability, efficiency, and ability to capture LRDs.

In the next few sections, we discuss the strategies employed to design a performant deep RNN architecture, which can achieve performance comparable to SSMs on LRA tasks.

### 5.2.1 Linear Recurrences

| Recurrence | sCIFAR | ListOps | Text | Retrieval |
|---|---|---|---|---|
| RNN-ReLU | 69.7 (0.2) | 37.6 (8.0) | 88.0 (0.1) | 88.5 (0.1) |
| RNN-Tanh | 69.9 (0.3) | 43.9 (0.1) | 87.2 (0.1) | 88.9 (0.2) |
| RNN-Lin | **72.2** (0.2) | **50.4** (0.2) | **89.1** (0.1) | **89.1** (0.1) |

Figure 5.2: Effect of removing the non-linearity from the recurrence on the feasible LRA tasks.

[1] parallelizability of Linear RNNs stem from the fact that they belong to the family of LTI SSMs

The first key step is to linearize the recurrence, a seemingly counterintuitive move given the widespread belief that non-linearities within the recurrence are crucial for RNNs to model complex functions. However, as evident from 5.2, in the context of deep architectures interleaved with non-linear MLPs, dropping the non-linearity in the recurrence actually boosted performance on LRA tasks [17]. This seemingly paradoxical finding aligns with the theoretical universality result [14] and the observation that non-linear projections applied after the linear RNN can be expressive, enabling the model to capture complex non-linear dynamics.

By switching to linear recurrences, LRUs gain several advantages:

- **Explicit Gradient Control**: The removal of non-linearities allows for direct control over how quickly gradients might vanish or explode. This is crucial for learning LRDs, as it enables careful tuning of the eigenvalues in the recurrent matrix to mitigate vanishing gradients.
- **Parallelizable Training**: Linear recurrences can be efficiently parallelized during training, leading to substantial speedups over non-linear RNNs which require sequential computation.

### 5.2.2 Complex Diagonal Recurrent Matrices

Building on the insights from S4 [7], LRUs also adopt the use of complex-valued diagonal recurrent matrices. This choice, while seemingly unusual for traditional RNNs, brings significant computational advantages and aligns with the theoretical findings on the benefits of complex numbers in linear recurrences [14].

#### Switching to Diagonal Matrices

Diagonal matrices offer substantial computational advantages due to their simplified structure. Let's examine how we can transform a dense linear RNN into its diagonal form without affecting expressivity.

We start with the recurrence of a linear RNN:

$$\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_k,$$

where $\mathbf{A} \in \mathbb{R}^{N\times N}$ is the recurrent matrix, $\mathbf{B} \in \mathbb{R}^{N\times D}$ is the input projection matrix, $\mathbf{x} \in \mathbb{R}^N$ is the hidden state, and $\mathbf{u} \in \mathbb{R}^D$ is the input. Assuming $\mathbf{x}_{-1} = 0$, we can unroll the recurrence as:

$$\mathbf{x}_k = \sum_{j=0}^{k-1} \mathbf{A}^j \mathbf{B}\mathbf{u}_{k-j}.$$

The exponentiation of $\mathbf{A}$ in this equation is the source of the vanishing/exploding gradient issue in RNNs. To better understand this, we leverage an eigenvalue analysis. Almost every matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ is diagonalizable[2], meaning we can write it as $\mathbf{A} = \mathbf{P}\mathbf{\Lambda}\mathbf{P}^{-1}$, where $\mathbf{P} \in \mathbb{C}^{N\times N}$ is an invertible matrix, and $\mathbf{\Lambda} = \text{diag}(\lambda_1, \lambda_2, ..., \lambda_N) \in \mathbb{C}^{N\times N}$ is a diagonal matrix containing the eigenvalues of $\mathbf{A}$.

[2]The set of non-diagonalizable matrices has measure zero

Note that, unlike symmetric matrices where eigenvalues and eigenvectors are real, non-symmetric matrices may have complex eigenvalues and eigenvectors.

Plugging this decomposition into the unrolled recurrence and multiplying both sides by $\mathbf{P}^{-1}$, we obtain:

$$\mathbf{P}^{-1}\mathbf{x}_k = \sum_{j=0}^{k-1} \mathbf{\Lambda}^j \mathbf{P}^{-1}\mathbf{B}\mathbf{u}_{k-j}.$$

By renaming $\overline{\mathbf{x}_k} \leftarrow \mathbf{P}^{-1}\mathbf{x}_k$ and $\overline{\mathbf{B}} \leftarrow \mathbf{P}^{-1}\mathbf{B}$, we arrive at a diagonal recurrence in the complex domain:

$$\overline{\mathbf{x}_k} = \sum_{j=0}^{k-1} \mathbf{\Lambda}^j \overline{\mathbf{B}}\mathbf{u}_{k-j}.$$

Finally, the output of the RNN can be computed as:

$$\mathbf{y}_k = \mathrm{Re}[\overline{\mathbf{C}}\mathbf{x}_k] + \mathbf{D}\mathbf{u}_k,$$

where $\overline{\mathbf{C}} = \mathbf{C}\mathbf{P}^{-1}$, and we take the real part of $\mathbf{C}\mathbf{x}_k$. Therefore, we can learn $(\mathbf{\Lambda}, \overline{\mathbf{B}}, \overline{\mathbf{C}}, \mathbf{D})$ instead of $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$, where $\mathbf{\Lambda}, \overline{\mathbf{B}}, \overline{\mathbf{C}}$ are complex-valued and $\mathbf{\Lambda}$ is diagonal.

This diagonal form allows for significant speedups both during training and inference as exponentiating a diagonal matrix simply involves exponentiating its diagonal elements, a trivial operation. Furthermore, unlike non-linear recurrences which need sequential computation, linear recurrences can be parallelized using efficient associative scans (as in S4 [7] and Mamba [6]).

### Controlling Stability with Eigenvalues

Recall that the diagonalized recurrence is given by: $\overline{\mathbf{x}_k} = \sum_{j=0}^{k-1} \mathbf{\Lambda}^j \overline{\mathbf{B}}\mathbf{u}_{k-j}$. Focusing on the $j$-th component of the hidden state at timestamp $k$, we observe its evolution as:

$$|x_{k,j}| = \mathcal{O}(|\overline{x}_{k-1,j}|) = \mathcal{O}(|\lambda_j|^k).$$

This equation clearly shows that the magnitude of the $j$-th eigenvalue, $|\lambda_j|$, directly determines the rate at which the signal from past inputs decays or grows in the corresponding hidden state dimension.

This leads us to the following observations:

- If $|\lambda_j| < 1$, the signal decays exponentially with each time step, leading to vanishing gradients and a limited ability to capture LRDs.
- If $|\lambda_j| > 1$, the signal grows exponentially, potentially leading to exploding gradients and numerical instability during training.

Therefore, to ensure stability and facilitate learning of LRDs, LRUs constrain the eigenvalues to lie within the unit circle, i.e., $|\lambda_j| < 1$ for all $j$. This constraint ensures that the influence of past inputs gradually decays, preventing exploding gradients while still allowing information from the distant past to contribute to the current state. This is exactly what the Glorot Equivalent Initialization achieves, as we see in the next section.

### Glorot Equivalent Initialization

To ensure a fair comparison with vanilla RNNs, Orvieto et al. [13] sought an initialization for the diagonal system that preserved the eigenvalue spectrum of a **Glorot-initialized dense RNN**. A Glorot initialized matrix [5] $\mathbf{A} \in \mathbb{R}^{N \times N}$ is such that each entry of $\mathbf{A}$ is sampled independently from a Gaussian with mean 0 and variance $\frac{1}{N}$. By leveraging the **Strong Circular Law**[3], we know that the eigenvalues of a Glorot-initialized dense RNN are uniformly distributed within the unit circle in the complex plane.

This leads to a simple yet effective initialization strategy for LRUs: sampling the diagonal elements of the recurrent matrix uniformly from the **unit disk in the complex plane**.

### Stable Exponential Parameterization

To further enhance stability and control over gradient flow, LRUs introduced a stable exponential parameterization for the diagonal elements of the recurrent matrix. This involved representing each diagonal element as the exponential of a complex number, decoupling magnitude and phase:

$$\lambda_j = \exp(-\exp(v_j^{log}) + i \exp(\theta_j^{log})),$$

where:

- $\exp(-\exp(v_j^{log}))$ is the magnitude, controlled by the learnable parameter $v_j^{log}$.
- $\exp(\theta_j^{log})$ is the phase, controlled by the learnable parameter $\theta_j^{log}$.

**Optimization under Exponential Parameterization** This exponential parameterization offers several advantages:

- **Decoupled Magnitude and Phase**: It decouples the learning of magnitude and phase, making optimization easier and more stable, especially when using optimizers like Adam.
- **Explicit Stability Control**: The exponential form ensures that the magnitude of each eigenvalue is always less than 1[4], guaranteeing stability and mitigating exploding gradients. To further enforce stability and allow for exploration of **initializations closer to the unit circle**, LRUs employ an additional exponential non-linearity on the magnitude, hence the double exponential. By initializing the eigenvalues closer to the unit circle, LRUs can be biased towards capturing longer-range interactions, further improving their performance on LRD tasks.
- **Reducing Eigenvalue Phase at Initialization**: For very long sequences, initializing the eigenvalues with a uniform phase distribution[5], can lead to

[3]Refer to **Theorem 3.1** of LRU paper [13] for more details

[4]$0 < e^{-|x|} \leq 1$ for all $x \in \mathbb{R}$

[5]This means uniformly sampling from $[0, 2\pi]$

the model learning spurious high-frequency oscillations, hindering its ability to capture global patterns. To address this, LRUs restrict the phase of the eigenvalues at initialization to a small range around 0, encouraging the model to learn more global, low-frequency features relevant for long-range reasoning. This can be done by initializing the phase parameter as $\exp(\theta_j^{log})$, which can ensure small phases by initializing the learnable $\theta_j^{log}$ to be negative.

To address the challenges of learning very long-range dependencies, especially those encountered in the PathX task, LRUs incorporate another key modification:

**Normalization**

As the eigenvalues of the recurrent matrix approach the unit circle, the **norm of the hidden state can potentially explode** during the forward pass[6]. To counteract this, LRUs introduced a normalization factor $\gamma \in \mathbb{R}^N$ for each dimension, initialized element-wise as $\gamma_i \leftarrow \sqrt{1 - |\lambda_i|^2}$, which modifies the recurrence as:

$$\mathbf{x}_k = \mathbf{\Lambda}\mathbf{x}_{k-1} + \gamma \odot \mathbf{B}\mathbf{u}_k,$$

where $\odot$ represents element-wise multiplication.

### 5.2.3 Deep LRU Architecture

The deep LRU architecture mirrors the general structure of deep SSMs and Transformers, incorporating residual connections, layer normalization, and non-linear mixing layers. The core difference lies in replacing the SSM or attention block with an LRU layer, parameterized as follows:

$$\mathbf{x}_k = \mathbf{diag}(\lambda)\mathbf{x}_{k-1} + \gamma \odot \mathbf{B}\mathbf{u}_k,$$

where:

- $\lambda_j = \exp(-\exp(v_j^{log})) + i\exp(\theta_j^{log}))$ are the diagonal elements of the recurrent matrix, parameterized using the stable exponential form with enforced stability.
- $\gamma_j = \sqrt{1 - |\lambda_j|^2}$ are the normalization factors, ensuring stability during the forward pass.
- $\mathbf{B}$ is the input projection matrix, mapping the input to the hidden state.

### 5.2.4 Empirical Results on LRA

With these carefully designed modifications, LRUs achieved impressive performance on the LRA benchmark [17], comparable to the S4 Models [7] as evident from 5.3. LRUs offer a computationally efficient alternative to SSMs and Transformers, especially for tasks requiring the processing of long sequences. Note that for this comparison, they used networks with depth of 6 residual blocks, with each

[6]For more details check **Proposition 3.3 Forward-pass blow-up** in the LRU paper [13]. This says that as we approach the unit circle in the limit, $\mathbb{E}|x_k|^2 \xrightarrow{\infty} \sum_{i=0}^{\infty} |\lambda|^{2i} = \frac{1}{1-|\lambda|^2}$

| | sCIFAR | ListOps | Text | Retrieval | Pathfinder | PathX |
|---|---|---|---|---|---|---|
| Linear Dense RNN | 72.2 (0.2) | 50.4 (0.2) | 89.1 (0.1) | 89.1 (0.1) | ✗ | ✗ |
| Diagonal Complex RNN | 86.5 (0.1) | 58.8 (0.3) | 87.4 (0.3) | 87.8 (0.5) | ✗ | ✗ |
| Stable Exp Param w/ Ring Init | 88.1 (0.0) | 59.4 (0.3) | 89.4 (0.1) | 90.1 (0.1) | 94.4 (0.3) | ✗ |
| $[r_{\min}, r_{\max}]$ | [0.9, 0.99] | [0.0, 1.0] | [0.0, 0.9] | [0.5, 0.9] | [0.9, 0.999] | |
| $+\gamma$ Normalization **(LRU)** | 89.0 (0.1) | 60.2 (0.8) | 89.4 (0.1) | 89.9 (0.1) | 95.1 (0.1) | 94.2 (0.4) |
| $[r_{\min}, r_{\max}]$ | [0.9, 0.999] | [0.0, 0.99] | [0.5, 0.9] | [0.5, 0.9] | [0.9, 0.999] | [0.999, 0.9999] |
| S4D (our reproduction) | 91.5 (0.2) | 60.2 (0.3) | 86.4 (0.0) | 89.5 (0.0) | 94.2 (0.3) | 97.5 (0.0) |
| S5 (our reproduction) | 88.8 (0.1) | 58.5 (0.3) | 86.2 (0.1) | 88.9 (0.0) | 95.7 (0.1) | 96.0 (0.1) |
| S4 (paper results) | *91.1* | *59.6* | *86.8* | *90.9* | *94.2* | *96.4* |
| S4D-LegS (paper results) | 89.9 | 60.5 | 86.2 | 89.5 | 93.1 | 91.9 |
| S5 (paper results) | *90.1* | *62.2* | *89.3* | *91.4* | 95.3 | *98.6* |

Figure 5.3: Effects of normalization on linear diagonal RNNs with stable exponential parameterization. Results showcase the advantage of taking initialization close to the unit circle under proper $\gamma$ normalization.

block consisting of identity skip connection, and the residual path containing a normalization layer[7], followed by the RNN/SSM block.

LRUs showcase the effectiveness of systematically analyzing and improving traditional RNNs, guided by insights from both theoretical analysis and the success of SSMs. They offer a powerful and computationally efficient alternative to attention, especially for tasks demanding the processing of long sequences.

[7]They used Batch Normalization for all

# Chapter 6

# Hybrid Models: Hawk and Griffin

Building upon the success of LRUs, De et al. [2] introduced Hawk, a pure RNN model, and Griffin, a hybrid model that combines gated linear recurrences with local attention. These models further push the boundaries of efficient long-range sequence modelling, achieving performance comparable to Transformers [20] while maintaining linear scaling in sequence length and exhibiting faster inference speeds.

## 6.1 Model Architecture with Residual Blocks

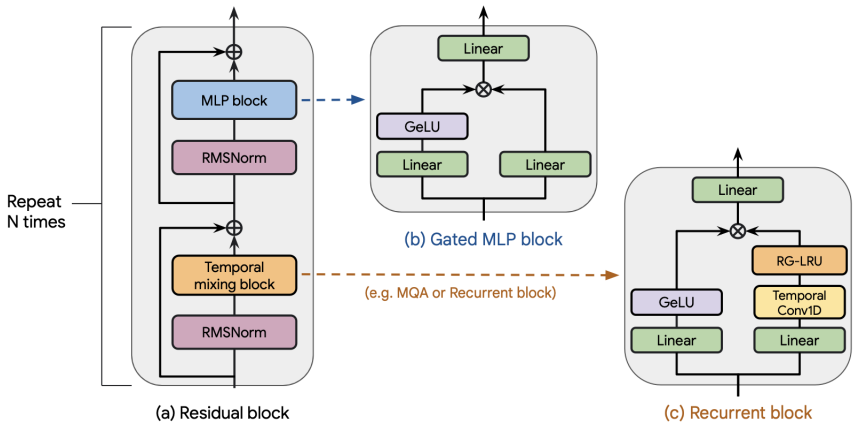



Figure 6.1: (a) The main backbone of Hawk and Griffin is the residual block, which is stacked $N$ times. (b) The gated MLP Block (c) The Recurrent Block containing the RG-LRU layer. For baseline comparison, this recurrent block is replaced with Multi-Query Attention (MQA) [16].

Both Hawk and Griffin adopt a similar architecture to Transformers and deep LRUs, employing residual blocks as their main building blocks. These residual blocks, shown in 6.1(a) of the Griffin paper [2], consist of:

- **RMSNorm**: Instead of Layer Normalisation, Hawk and Griffin utilize Root Mean Square Layer Normalization (RMSNorm) by Zhang et al. [19], which has been empirically beneficial for Transformer-based language models.

- **Temporal Mixing Block**: As illustrated in 6.1(c), this is the core component that distinguishes Hawk and Griffin. They replace the MQA/MHA block in Transformers with either a recurrent block (for Hawk) or a combination of recurrent and local attention blocks (for Griffin).
- **Gated MLP Block**: Similar to Transformers, a gated MLP block is applied to each token's representation after the temporal mixing block. The details of this block are illustrated in 6.1(b).

## 6.2 RG-LRU: A Modification of LRU

The recurrent block in Hawk and Griffin employs a modified version of the LRU, called the Real-Gated Linear Recurrent Unit (RG-LRU). This modification introduces gating mechanisms to the LRU, enhancing its ability to selectively propagate or forget information based on the input. The RG-LRU equations are as follows:

$$\begin{aligned}
\mathbf{r}_t &= \sigma(\mathbf{W}_a \mathbf{x}_t + \mathbf{b}_a), \text{ recurrence gate} \\
\mathbf{i}_t &= \sigma(\mathbf{W}_x \mathbf{x}_t + \mathbf{b}_x), \text{ input gate} \\
\mathbf{a}_t &= \mathbf{a}^{c\mathbf{r}_t}, \\
\mathbf{h}_t &= \mathbf{a}_t \odot \mathbf{h}_{t-1} + \sqrt{1 - \mathbf{a}_t^2} \odot (\mathbf{i}_t \odot \mathbf{x}_t).
\end{aligned}$$

The output of the layer is $\mathbf{y}_t = \mathbf{h}_t$, and the sigmoid function is denoted by $\sigma$. The recurrent weight $\mathbf{a}$ is diagonal, allowing for element-wise operations. To ensure stability, $\mathbf{a}$ is parameterized as $\mathbf{a} = \sigma(\mathbf{\Lambda})$, where $\mathbf{\Lambda}$ is a learnable parameter, and $c$ is a scalar constant set to 8.

### 6.2.1 Gating mechanism of RG-LRU

The input gate, $\mathbf{i}_t$, is similar to that in LSTMs, which allows filtering (scaling down) the input $\mathbf{x}_t$. The recurrence gate, $\mathbf{r}_t$, is unique to the RG-LRU. While the selection mechanism in Mamba [6] interpolates between the previous hidden state and the current input, similar to the update gate in GRUs, the recurrence gate in RG-LRU can approximately interpolate between the standard LRU update and the previous state. This allows it to discard the input effectively and preserve information from the previous history. This capability is crucial for achieving super-exponential memory by reducing the influence of uninformative inputs.

### 6.2.2 Temporal Conv 1D Layer

Unlike the LRU, the RG-LRU layer incorporates a small depthwise 1D convolutional layer before the recurrence. This convolutional layer, inspired by the Shift-SSM in H3 [3], operates on a short temporal context (filter dimension of 4). It provides local context to the RG-LRU layer, enhancing its ability to capture short-term dependencies. The output of the convolutional layer is then fed into the RG-LRU layer, as illustrated in 6.1(c).

## 6.3 Local Sliding Window Attention

Griffin utilizes local attention, also known as sliding window attention, to further improve its ability to model sequences. This local attention mechanism allows each token to attend only to a fixed number of tokens within a local window, reducing computational complexity compared to global attention while still capturing relevant contextual information. The combination of local attention with the global receptive field of the RG-LRU provides a powerful and efficient mechanism for sequence modelling.

## 6.4 Hawk

Hawk is a pure RNN model that solely relies on RG-LRU blocks for temporal mixing. Its architecture comprises stacking multiple residual blocks, each containing an RG-LRU layer followed by a gated MLP block and residual connections.

## 6.5 Griffin

Griffin combines the strengths of RG-LRU and local attention. Its architecture alternates between two residual blocks with an RG-LRU layer and one residual block with a local sliding window attention layer. This layered structure enables Griffin to capture both long-range dependencies (through RG-LRU) and local context (through local attention), leading to improved performance and efficiency.

## 6.6 Optimisations

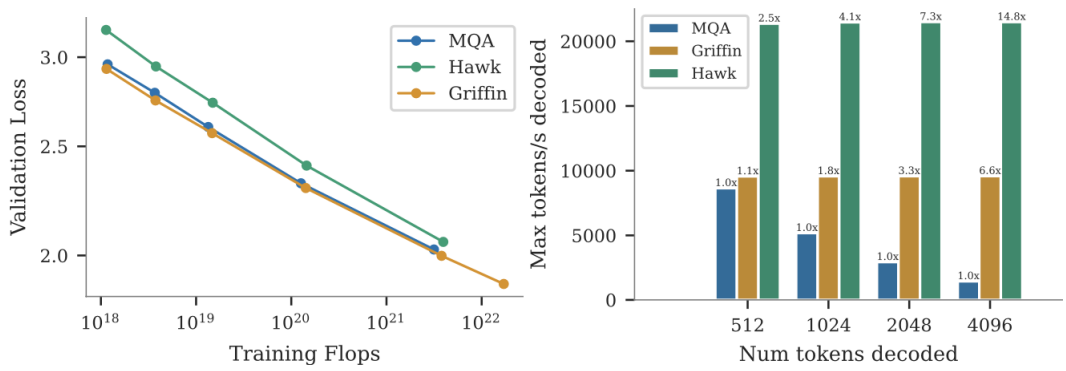
To further enhance computational efficiency, especially on modern hardware, Griffin incorporates a custom linear scan implementation for the RG-LRU layer (similar to Mamba [6]). This optimisation is crucial because diagonal RNNs, like RG-LRU, have a low FLOPs-to-byte ratio, making them memory-bound on current accelerators. The custom linear scan minimises memory transfers by keeping the hidden state in the fast on-chip memory (SRAM) throughout the computation.

## 6.7 Empirical Results

Both Hawk and Griffin have achieved remarkable results on language modelling tasks, showcasing their potential as efficient alternatives to Transformers:

### 6.7.1 Scaling Laws and Throughput

As evident from 6.2(a), Hawk, Griffin and MQA baseline exhibit power law scaling between validation loss and training FLOPs, with Griffin achieving the lowest validation loss. In terms of throughput, both Griffin and Hawk thoroughly outperform MQA, especially as the length of the sample increases, as shown in 6.2(b).




(a) Scaling curve during training (b) Maximum throughput at 1B parameter scale.

Figure 6.2: Comparing validation loss and throughput of Hawk, Griffin and MQA on custom dataset.

### 6.7.2 Character Normalised Accuracy

| Model Type | Model Size | Training Tokens | MMLU | HellaSwag | PIQA | WinoGrande | ARC-E | ARC-C | Average |
|---|---|---|---|---|---|---|---|---|---|
| **Mamba** | 3B | 600B | 26.2 | 71.0 | 78.1 | 65.9 | 68.2 | 41.7 | 58.5 |
| **Llama-2** | 7B | 2T | 45.3 | 77.2 | 78.8 | 69.2 | 75.2 | 45.9 | 65.3 |
| | 13B | 2T | **54.8** | 80.7 | 80.5 | 72.8 | 77.3 | 49.4 | 69.3 |
| **MQA Transformer (Baseline)** | 1B | 300B | 28.9 | 64.8 | 75.0 | 62.0 | 60.2 | 35.4 | 54.4 |
| | 3B | 300B | 31.7 | 71.0 | 77.6 | 66.1 | 68.1 | 39.2 | 59.0 |
| | 6B | 300B | 38.9 | 77.0 | 79.5 | 70.4 | 74.1 | 45.2 | 64.2 |
| **Hawk** | 1B | 300B | 29.7 | 63.3 | 76.1 | 57.2 | 60.6 | 34.6 | 53.6 |
| | 3B | 300B | 31.3 | 71.7 | 78.8 | 66.5 | 68.4 | 40.2 | 59.5 |
| | 7B | 300B | 35.0 | 77.6 | 80.0 | 69.9 | 74.4 | 45.9 | 63.8 |
| **Griffin** | 1B | 300B | 29.5 | 67.2 | 77.4 | 65.2 | 67.0 | 36.9 | 57.2 |
| | 3B | 300B | 32.6 | 73.5 | 78.1 | 67.2 | 71.5 | 41.4 | 60.7 |
| | 7B | 300B | 39.3 | 78.6 | 81.0 | 72.6 | 75.4 | 47.9 | 65.8 |
| | 14B | 300B | 49.5 | **81.4** | **81.8** | **74.1** | **79.1** | **50.8** | **69.5** |

Figure 6.3: Comparison of character normalized accuracy of different models of different sizes on various downstream tasks.

As shown by this table at 6.3, the character-normalised accuracy of Hawk and Griffin on various downstream tasks. Hawk exceeds the performance of Mamba [6], while Griffin achieves comparable accuracy to LLaMA-2 [18], despite being trained on significantly fewer tokens. Note that the pre-training datasets used here are different.

These results, combined with their efficiency advantages in training and inference, position Hawk and Griffin as compelling alternatives to attention-based models.

# Chapter 7

# Experiments

In this chapter, we present some results from the experiments conducted using the Transformer and Mamba models on three different datasets: Tiny Shakespeare, Maestro V2, and Sequential MNIST. The experiments aimed to compare the performance of the two models on diverse tasks, including character-level language modelling, MIDI generation, and sequential digit generation.

Here is a brief over of the models used in the experiments:

- **Transformer**: The Transformer model used in the experiments is based on GPT-2 architecture [15], inspired by the nanoGPT repository by Andrej Karpathy. It consists of a stack of Decoder Blocks, each containing a Multi-Head Self-Attention layer and a MLP layer. Due to limited compute, we stuck to a smaller model with $2.4M$ parameters. We fix a cosine decay learning rate schedule, starting from $10^{-3}$ and decaying to $10^{-4}$ over the course of training.

- **Mamba**: Due to limited resources, we used a smaller model with $2.2M$ parameters for Mamba as well. For Mamba we use 10 times smaller learning rate than the Transformer model. So, we use a cosine decay learning rate schedule, starting from $10^{-4}$ and decaying to $10^{-5}$ over the course of training.

Note, we could not experiment with Hawk and Griffin, as they require more modern GPUs supporting newer versions of Triton.

## 7.1 Tiny Shakespeare

The Tiny Shakespeare dataset [11] consists of 40,000 lines of text from various plays by William Shakespeare. The task is to train a character-level language model to generate text similar to Shakespeare's writing style. Here, we use a vocabulary of 65 characters and a context length of 256 tokens.

As evident from 7.1, Mamba seems to converge faster than the Transformer model, even though it has a lower learning rate, but it ends up overfitting the data. The generated text samples in 7.2 show that both models are able to generate text that resembles Shakespeare's writing style from scratch.

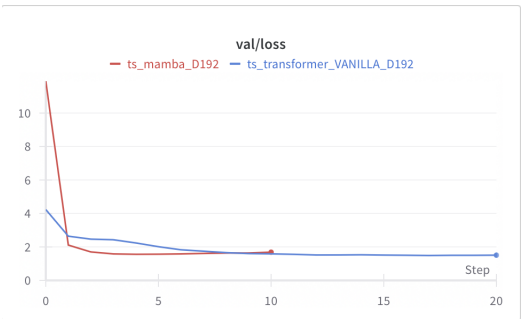



Figure 7.1: Tiny Shakespeare: Loss Curves

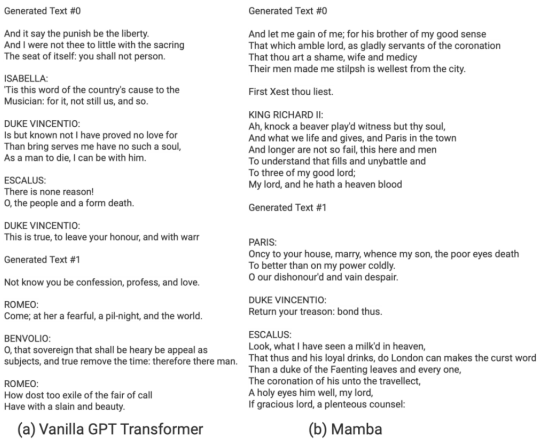



Figure 7.2: Generated text samples

## 7.2 Maestro V2

The Maestro V2 dataset [10] contains about 200 hours of paired audio and MIDI recordings from ten years of International Piano-e-Competition. The task is to train a MIDI generating model that can generate music similar to the pieces in the dataset. We first preprocess the MIDI files using the MidiTok library to tokenize the MIDI files using a BPE tokenizer. Here, we use a vocabulary of 8000 tokens and a context length of 256 tokens.

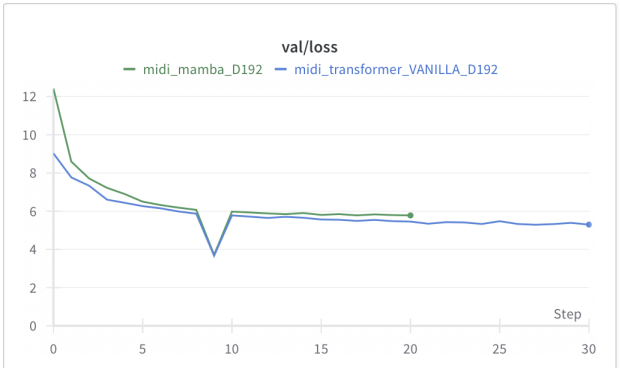



Figure 7.3: MIDI: Loss Curves

Repeating the trend, Mamba converges faster than the Transformer model, as seen in 7.3, even though it has a lower learning rate. But it still ends up overfitting the data. The generated MIDI samples from both models are not shown here, but they are available in the code repository.

## 7.3 Sequential MNIST

The Sequential MNIST dataset is a modified version of the MNIST dataset [12], where we quantize the pixel values to 3-bit pixels with values from $[0, 7]$. We reshape the $28 \times 28$ images to $16 \times 16$ and flatten them to a 256-dimensional array. We can flatten the images using various strategies, but from empirical observations the best results were obtained using the identity flattening strategy,

where we flatten the image from left to right, one row after another. We train our models only on the digit 0 to make the task easier and converge faster.

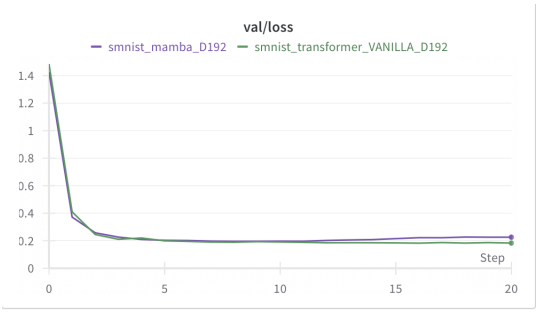



Figure 7.4: SMNIST: Loss Curves

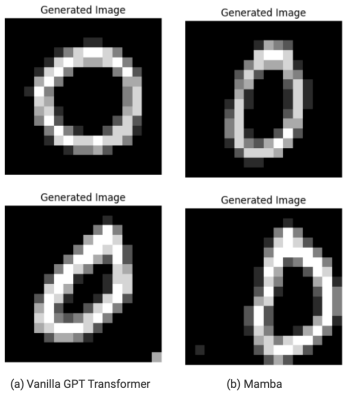



Figure 7.5: Generated digit samples

As seen in 7.4, Mamba ends up slightly overfitting on the data again. The generated digit samples from both the models are shown in 7.5. Both models are able to sequentially generate digits pixel-by-pixel from scratch.

# Bibliography


[1] Tri Dao et al. *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. 2022. arXiv: 2205.14135 [cs.LG].

[2] Soham De et al. *Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models*. 2024. arXiv: 2402.19427 [cs.LG].

[3] Daniel Y. Fu et al. *Hungry Hungry Hippos: Towards Language Modeling with State Space Models*. 2023. arXiv: 2212.14052 [cs.LG].

[4] Leo Gao et al. *The Pile: An 800GB Dataset of Diverse Text for Language Modeling*. 2020. arXiv: 2101.00027 [cs.CL].

[5] Xavier Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Journal of Machine Learning Research - Proceedings Track* 9 (Jan. 2010), pp. 249–256.

[6] Albert Gu and Tri Dao. *Mamba: Linear-Time Sequence Modeling with Selective State Spaces*. 2023. arXiv: 2312.00752 [cs.LG].

[7] Albert Gu, Karan Goel, and Christopher Ré. *Efficiently Modeling Long Sequences with Structured State Spaces*. 2022. arXiv: 2111.00396 [cs.LG].

[8] Albert Gu et al. *HiPPO: Recurrent Memory with Optimal Polynomial Projections*. 2020. arXiv: 2008.07669 [cs.LG].

[9] Ankit Gupta, Albert Gu, and Jonathan Berant. *Diagonal State Spaces are as Effective as Structured State Spaces*. 2022. arXiv: 2203.14343 [cs.LG].

[10] Curtis Hawthorne et al. "Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=r1lYRjC9F7.

[11] Andrej Karpathy. *char-rnn*. https://github.com/karpathy/char-rnn. 2015.

[12] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[13] Antonio Orvieto et al. *Resurrecting Recurrent Neural Networks for Long Sequences*. 2023. arXiv: 2303.06349 [cs.LG].

[14] Antonio Orvieto et al. *Universality of Linear Recurrences Followed by Nonlinear Projections: Finite-Width Guarantees and Benefits of Complex Eigenvalues*. 2024. arXiv: 2307.11888 [cs.LG].

[15] Alec Radford et al. "Language Models are Unsupervised Multitask Learners". In: (2019).

[16] Noam Shazeer. *Fast Transformer Decoding: One Write-Head is All You Need.* 2019. arXiv: 1911.02150 [cs.NE].

[17] Yi Tay et al. *Long Range Arena: A Benchmark for Efficient Transformers.* 2020. arXiv: 2011.04006 [cs.LG].

[18] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models.* 2023. arXiv: 2307.09288 [cs.CL].

[19] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models.* 2023. arXiv: 2302.13971 [cs.CL].

[20] Ashish Vaswani et al. *Attention Is All You Need.* 2023. arXiv: 1706.03762 [cs.CL].