Understanding the Development of State Space Models (SSMs) and their Performance in Language Applications

Manav Rajiv Moorthy
University of Washington
manavrm@uw.edu

Prateek Mahajan
University of Washington
pmahajan@uw.edu

1. Introduction

In the realm of Natural Language Processing (NLP), foundation models have emerged as transformative tools that underpin a vast array of language applications, ranging from machine translation and summarization to question-answering and content generation. These models, trained on extensive datasets and fine-tuned for specific tasks, leverage massive computational power and advanced architectures to achieve state-of-the-art results.

Transformer-based architectures have revolutionized the domain, and their success can be largely attributed to the introduction of a novel self-attention mechanism [12]. This mechanism allows models to capture contextual relationships between words irrespective of their distance in a sequence, enabling a deep understanding of syntax and semantics. Their capability of parallelized training and efficient inference gives transformers a significant advantage over traditional recurrent models.

While transformers excel in capturing dependencies and generating coherent text, they face computational inefficiencies when processing long sequences. The self-attention operation requires computing attention scores between all token pairs, resulting in a computational and memory cost that scales quadratically with the sequence length. As a result, transformers often require substantial hardware resources, limiting their practicality for resource-constrained environments or real-time applications.

Structured state space models (SSMs) have emerged as promising alternatives to address these limitations, offering efficient mechanisms to model long-range dependencies [1]. Unlike transformers, SSMs process sequences with linear computational complexity, making them inherently more suitable for applications requiring the analysis of long sequences from a systems performance perspective. Recent research has introduced advanced variants such as the S4, S5, H3, Mamba and Mamba-2 architectures, which extend the capabilities of traditional SSMs to achieve content-aware and context-aware reasoning effectively.

In this project, we compare the performance of SSMs against transformers in various scenarios, including few-shot learning, long sequence processing, and retrieval-augmented generation (RAG). We further run experiments on reasoning tasks such as Chain of Thought (CoT) prompting, to evaluate their relative adaptability and effectiveness in addressing diverse NLP challenges.

1.1. Research Questions

- What alternative architectures have been proposed recently to address current bottlenecks seen in transformers? What are the trade-offs of using these models versus transformers?
- How can systems be leveraged to achieve performance improvements by using the alternative models (specifically SSMs)?
- How efficient and effective are these SSM architectures in diverse language tasks, when compared to transformer-based architectures?

2. Related Works

SSMs represent any recurrent process with a latent state. They use a set of variables called state variables to encapsulate the system's internal state at any given time. These models are widely used in control theory, signal processing, time series analysis. More recently, they have inspired many machine learning architectures for sequence modeling tasks in language, audio and forecasting applications. A representation of the state space model can be viewed in Figure 1.

The state space model is defined by the simple equation below:

$$h_t = Ah_{t-1} + Bx_t \tag{1a}$$
$$y_t = Ch_t \tag{1b}$$

where 1-dimensional function or sequence $x(t) \in \mathbb{R} \mapsto y(t) \in \mathbb{R}$ through an implicit latent state $h(t) \in \mathbb{R}^N$.

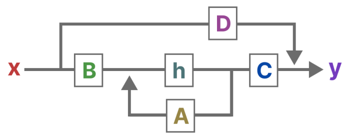


Figure 1. A flow diagram of a state space model. Here, *x* is the input, *h* is the hidden state, and *y* is the output. Matrices *A*, *B*, *C* and *D* help capture the state variables over time. *D* can be viewed as the equivalent of a skip connection in deep learning architectures, and are generally ignored in the mathematical expression of state space models.

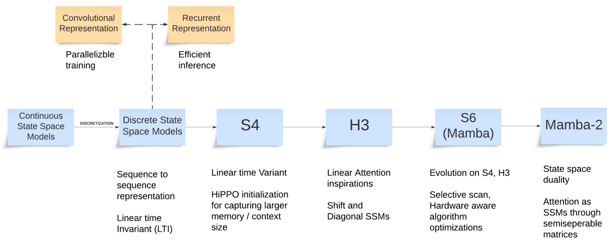

Convolutional Representation
Recurrent Representation
Parallelizble training
Efficient inference
Continuous State Space Models
DISCRETIZATION
Discrete State Space Models
S4
H3
S6 (Mamba)
Mamba-2
Sequence to sequence representation
Linear time Invariant (LTI)
Linear time Variant
HiPPO initialization for capturing larger memory / context size
Linear Attention inspirations
Shift and Diagonal SSMs
Evolution on S4, H3
Selective scan, Hardware aware algorithm optimizations
State space duality
Attention as SSMs through semiseperable matrices

Figure 2. A flow diagram of the evolution of SSM architectures over time.

SSMs are traditionally continuous models. However, they must be discretized to be compatible with sequence learning applications, whose inputs are discrete and tokenwise. Mathematical principles like zero-order hold (ZOH) can be used for discretization, enabling the conversion of continuous dynamics into discrete-time systems suitable for sequence processing.

A discretized form of the state space model is expressed below:

$$h_t = \bar{A}h_{t-1} + \bar{B}x_t \tag{2a}$$
$$y_t = \bar{C}h_t \tag{2b}$$

The matrices $\bar{A}$, $\bar{B}$ and $\bar{C}$ the discretized versions of A, B and C.

Structured state space sequence models (S4) are a recent class of sequence models for deep learning [6]. One interesting observation in S4 was that it was proposed to be an SSM that could be computed by using either a convolution or a recurrence. Parameterization in S4 efficiently swaps between these representations, allowing it to excel at long sequences. An illustration to understand the above is shown in Figure 3.

Concretely, S4 models are defined with four parameters (Δ, A, B, C), which define a sequence-to-sequence transformation in two stages. The set of equations 2 and 3 illustrate the computation of the SSM in recurrent and convolutional forms. Hence, S4 architecture perform both efficient auto-regressive inference (with the recurrent form, since they have a fixed memory that does not scale quadratically with sequence length as it does for transformers) and efficient training (with matrix multiplications in the convolution form).

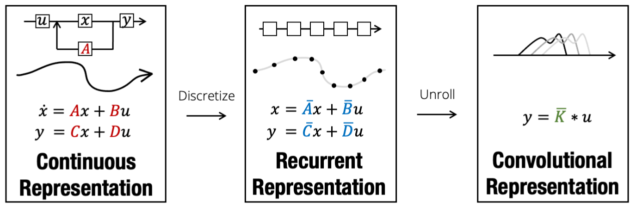

u
x
y
A
$\dot{x} = Ax + Bu$
$y = Cx + Du$
Continuous Representation
Discretize
$x = \bar{A}x + \bar{B}u$
$y = \bar{C}x + \bar{D}u$
Recurrent Representation
Unroll
$y = \bar{K} * u$
Convolutional Representation

Figure 3. The convolutional and recurrent representations of S4 after discretization.

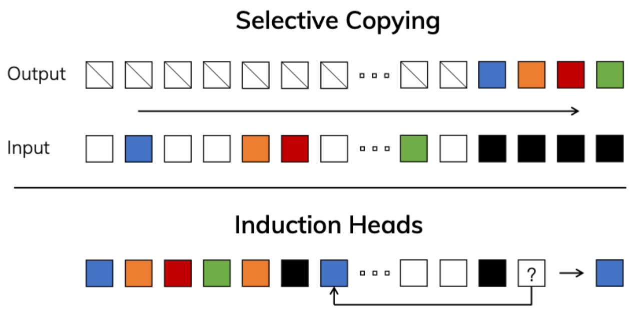


Figure 4. One task is Selective Copying, a modification of regular Copying, where the distance between remembered tokens can vary, and models need to selectively remember or ignore input depending on its content. The other is Induction Heads from Transformer Circuits, involving prefix matching within the context followed by copying. LTI systems fail these tasks.

$$\bar{K} = \left(CB, CAB, \dots, CA^kB, \dots\right) \tag{3a}$$
$$y = x * \bar{K} \tag{3b}$$

However, while the formulation of S4 enables efficient handling of long term dependencies, they face limitations as they operate as a Linear Time-Invariant (LTI) system. This behavior restricts their ability to capture nonlinear relationships, reducing their expressive power for complex data. Additionally, the memory structure is fixed by the parameters $\bar{A}$, $\bar{B}$ and $\bar{C}$, which may not adapt well to diverse temporal patterns. Furthermore, ensuring stability during training is another challenge, as large-scale or high-dimensional data can destabilize the system.

LTI systems, such as S4, struggle with tasks like selective copying and induction head tasks due to inherent limitations in their design and operational characteristics. The tasks are explained in Figure 4.

H3 extends S4 by introducing hybrid SSMs that incorporate multiplicative interactions and discrete operations [3]. Their system can be described via the set of equations 4.

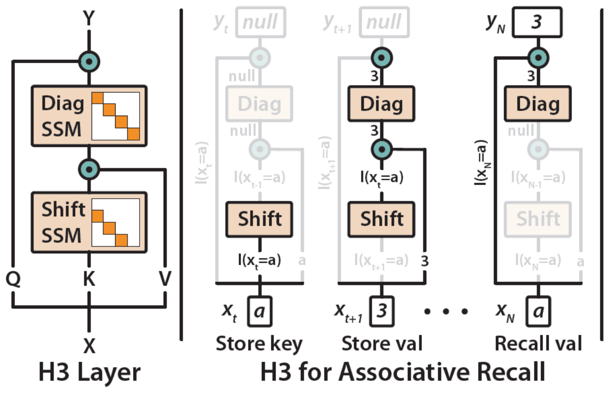
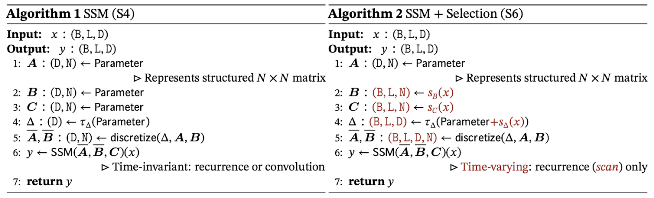


Figure 5. The illustration of shift and diagonal SSMs in H3.



```
Algorithm 1 SSM (S4)
Input: $x$ : (B, L, D)
Output: $y$ : (B, L, D)
1: $A$ : (D, N) ← Parameter
                    ▷ Represents structured $N \times N$ matrix
2: $B$ : (D, N) ← Parameter
3: $C$ : (D, N) ← Parameter
4: $\Delta$ : (D) ← $\tau_\Delta$(Parameter)
5: $\overline{A}, \overline{B}$ : (D, N) ← discretize($\Delta, A, B$)
6: $y$ ← SSM($\overline{A}, \overline{B}, C$)($x$)
                    ▷ Time-invariant: recurrence or convolution
7: return $y$
```

```
Algorithm 2 SSM + Selection (S6)
Input: $x$ : (B, L, D)
Output: $y$ : (B, L, D)
1: $A$ : (D, N) ← Parameter
                    ▷ Represents structured $N \times N$ matrix
2: $B$ : (B, L, N) ← $s_B(x)$
3: $C$ : (B, L, N) ← $s_C(x)$
4: $\Delta$ : (B, L, D) ← $\tau_\Delta$(Parameter$+s_\Delta(x)$)
5: $\overline{A}, \overline{B}$ : (B, L, D, N) ← discretize($\Delta, A, B$)
6: $y$ ← SSM($\overline{A}, \overline{B}, C$)($x$)
                    ▷ Time-varying: recurrence (scan) only
7: return $y$
```

Figure 6. The difference in algorithms for S4 and S6 blocks in SSMs.

$$K_t = \text{SSM}_{\text{shift}}(K) \odot V \tag{4a}$$

$$x_t = \text{SSM}_{\text{diag}}(K_t) \tag{4b}$$

$$y_t = Q \odot x_t \tag{4c}$$

These components are further explained below:

1. SSM_{shift}: A shift matrix captures temporal dynamics by shifting token states.

2. SSM_{diag} : A diagonal matrix to retain long-term memory.

3. Q, K, V: Learnable projections inspired by attention mechanisms.

The H3 block is inspired by linear attention and aims at mimicking its functionality in a more efficient manner [7]. As illustrated in Figures 5 and 7, the H3 architecture incorporates a ”conv” operation (denoted as SSM_{shift} in Figure 5) to capture temporal dynamics by applying a shift matrix to the key (K) of the input. This operation encodes interactions between adjacent token states. The resulting output is then multiplied with the value (V) to form a matrix representing token interactions. This matrix is subsequently processed through SSM_{diag}, which condenses the information further. Finally, the query (Q) is applied to the output of SSM_{diag}, selectively extracting the most relevant interactions from the condensed information matrix.

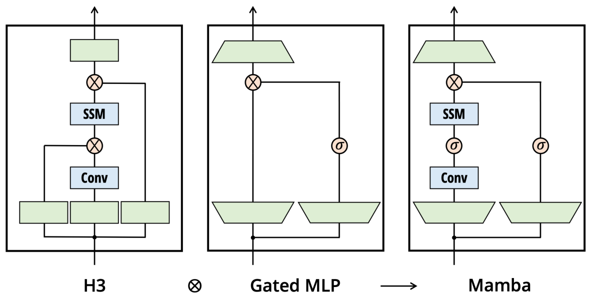


Figure 7. Architectural evolution from H3 to Mamba. H3 leverages hybrid SSMs with linear and nonlinear components, while Mamba introduces additional modules for dynamic state adaptation and improved scalability.

While H3 is inspired by linear attention interleaved with an MLP (multi-layer perceptron) block, the authors of Mamba drew inspiration from the designs of Gated Attention Units (GAU) [5]. Mamba proposed a new S6 variant by leveraging selective state spaces, which allow it to dynamically focus on relevant token interactions by parameterizing the state variables with the input data.

Mamba simplifies and unifies H3 by converting its multiplicative interactions into activation functions. It further modifies the SSM layer used in the H3 block to an S6 layer instead of S4 to improve H3's ability to use context while generating output. This has been described further in Figure 6. This, however, prevents the SSM to be operable in convolutional mode. To address this, Mamba also employs hardware-aware parallel algorithms in recurrent mode optimized for modern accelerators. The structure of the Mamba block is illustrated in Figure 7.

Moreover, Mamba reduces the number of SSM parameters (Δ, A, B, C) compared to H3, further enhancing its computational efficiency. Mamba saw widespread adoption of the idea of selective state spaces in various domains - vision [14] [9] [8], genomics [11], graph-based sequence modeling [13], and in-context learning [4] [10]. These works highlighted the versatility and potential of SSMs as a unified framework for sequence modeling, bridging paradigms such as convolutional, recurrent, and attention-based models.

Despite Mamba's advancements, its connection to attention mechanisms remained unclear, and its selective scan implementation, though efficient, lacked the hardware-optimized benefits of attention for training. Mamba-2 addresses these challenges by introducing Structured State Space Duality (SSD), a novel SSM variant that unifies the strengths of SSMs and attention while improving computational efficiency [2].

The SSD layer is designed as a standalone component supported by an algorithm that allows for faster computation compared to earlier SSM implementations. This duality framework enhances the theoretical understanding of SSMs and makes Mamba-2 a practical and scalable solution for large-scale sequence modeling tasks.

3. Proposed Methodology

Recent works on SSMs have demonstrated their potential across a variety of diverse tasks, yet they remain minimally explored in language applications. Notably, these studies primarily evaluated SSMs in zero-shot settings, leaving their efficacy in more complex and diverse NLP tasks underexplored.

While SSMs are known for their computational efficiency due to their ability to condense sequence information into a single hidden state, the key question we aim to address is whether this efficiency translates into effectiveness across diverse NLP tasks. We propose to explore the performance of SSMs in four experiments, each targeting a specific linguistic or reasoning capability. These experiments focus on testing SSMs beyond their traditionally limited zero-shot evaluations, addressing specific objectives and challenges that SSMs must overcome to be viable alternatives to transformers in real-world applications.

3.1. Few-Shot Prompting

Objective: To evaluate the adaptability and performance of SSMs in few-shot prompting scenarios.

Rationale: Few-shot prompting is critical in NLP applications where labeled data is scarce, such as low-resource languages or domain-specific tasks. Few-shot prompting tests whether SSMs can infer the task's objective and learn patterns from a small number of examples in the input prompt. This setup allows us to directly compare the performance of SSMs, designed for efficient sequence modeling, with transformers that are known for excelling in such tasks.

3.2. Handling Long(-ish) Contexts

Objective: To test the ability of SSMs to handle moderately long sequences efficiently while maintaining comprehension and relevance in outputs.

Rationale: Long-range sequence modeling is a known challenge for transformers due to their quadratic complexity in self-attention. SSMs, designed for efficient sequential processing, are theoretically better suited for tasks involving extended contexts. This setup evaluates whether SSMs can maintain context coherence and relevance while scaling efficiently for longer sequences. Due to resource constraints (we used one available GPU at a time in the AMD cluster provided), we had compute and memory limitations that needed to be taken into consideration. Hence, we term this experiment as a long-ish context experiment as we ran the experiment on moderately long sequences.

3.3. Retrieval Augmented Generation (RAG)

Objective: To assess the ability of SSMs as a generator to integrate retrieved context into sequence generation effectively.

Rationale: RAG systems are crucial for open-domain question answering, knowledge-based reasoning, and content generation. Transformers, with their quadratic self-attention complexity, excel at such tasks by modeling all pairwise interactions between tokens, enabling them to extract and integrate relevant context effectively. In contrast, SSMs condense sequence information into a single hidden state representation, bypassing the need for explicit token-to-token interactions. This experiment is crucial for assessing whether SSMs can balance efficient context processing with coherent output generation in scenarios requiring retrieval and synthesis of external knowledge.

3.4. Chain of Thought (CoT) Prompting

Objective: To evaluate the reasoning and intermediate step-by-step processing capabilities of SSMs.

Rationale: CoT prompting is essential for tasks requiring multi-step reasoning, such as arithmetic problem-solving or logical deduction. This setup determines whether SSMs can adapt to reasoning-heavy NLP tasks.

4. Expected Outcome

We will provide a comprehensive effectiveness analysis of the Mamba and Mamba-2 architectures on the AMD MI2104X accelerator. By benchmarking these models across diverse language application scenarios mentioned above, we aim to generate actionable insights into the operational efficiency and understand the effectiveness of SSMs in language tasks. Comparisons with traditional transformer-based architectures will further highlight their relative strengths and limitations, offering valuable guidance for their potential deployment in various NLP applications.

5. Challenges

Firstly, developing a deep understanding of the pipeline, theory, and intuition behind the Mamba and Mamba-2 architectures is non-trivial. These architectures build upon advanced mathematical and system principles such as control theory.

Second, implementing and running the provided GitHub repositories for these models on AMD MI2104X GPUs presents a technical hurdle. Most existing codebases for the SSM architectures mentioned above are optimized for CUDA, with limited support or testing on AMD hardware.

This necessitates exploring compatibility layers or adapting the code to effectively leverage AMD's GPU ecosystem, potentially requiring additional debugging and performance tuning.

Finally, identifying the scope for optimization poses a significant challenge. The level of optimization detailed in the original research papers of Mamba and Mamba-2 is highly sophisticated, leaving little room for improvement without extensive expertise and computational resources. Pinpointing bottlenecks or potential enhancements while ensuring fidelity to the original architectures adds an additional layer of complexity to this undertaking.

6. Experiments and Results

7. Key Takeaways

8. Future Works

References

[1] Carmen Amo Alonso, Jerome Sieber, and Melanie N. Zeilinger. State space models as foundation models: A control theoretic overview, 2024. 1

[2] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality, 2024. 3

[3] Daniel Y. Fu, Tri Dao, Khaled K. Saab, Armin W. Thomas, Atri Rudra, and Christopher Ré. Hungry hungry hippos: Towards language modeling with state space models, 2023. 2

[4] Riccardo Grazzi, Julien Siems, Simon Schrodi, Thomas Brox, and Frank Hutter. Is mamba capable of in-context learning?, 2024. 3

[5] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024. 3

[6] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *CoRR*, abs/2111.00396, 2021. 2

[7] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. *CoRR*, abs/2006.16236, 2020. 3

[8] Yue Liu, Yunjie Tian, Yuzhong Zhao, Hongtian Yu, Lingxi Xie, Yaowei Wang, Qixiang Ye, and Yunfan Liu. Vmamba: Visual state space model, 2024. 3

[9] Jun Ma, Feifei Li, and Bo Wang. U-mamba: Enhancing long-range dependency for biomedical image segmentation, 2024. 3

[10] Jongho Park, Jaeseung Park, Zheyang Xiong, Nayoung Lee, Jaewoong Cho, Samet Oymak, Kangwook Lee, and Dimitris Papailiopoulos. Can mamba learn how to learn? a comparative study on in-context learning tasks, 2024. 3

[11] Yair Schiff, Chia-Hsiang Kao, Aaron Gokaslan, Tri Dao, Albert Gu, and Volodymyr Kuleshov. Caduceus: Bi-directional equivariant long-range dna sequence modeling, 2024. 3

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. 1

[13] Chloe Wang, Oleksii Tsepa, Jun Ma, and Bo Wang. Graph-mamba: Towards long-range graph sequence modeling with selective state spaces, 2024. 3

[14] Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. Vision mamba: Efficient visual representation learning with bidirectional state space model, 2024. 3