Mamba–the First Linear-time Sequence Model that Achieves Transformer-quality Performance

Xiaoke Song (xsong40), **Ruoyun Yang** (ryang104), **Kejing Yan** (kyan28)
DSI, Brown University
https://github.com/anyfruit/mamba-sentiment-dataset.git[1]

1 Introduction

In this project, we re-implement the Mamba model (Mamba: Linear-Time Sequence Modeling with Selective State Spaces, Gu et al., 2024)[3] in TensorFlow and evaluate its performance on the Sentiment140 dataset [2], which consists of 1.6 million tweets labeled as positive, negative, or neutral. Sentiment analysis is a classic NLP task that challenges models to understand context, sarcasm, and subtle shifts in meaning, making it ideal for testing sequence models. Our focus is to assess Mamba's efficiency and effectiveness on textual data, and to compare it directly against Transformer models to evaluate potential trade-offs between accuracy and computational cost.

1.1 Motivation

We selected Mamba for its innovative approach to sequential modeling, which builds on structured state-space model (SSMs) like S4 (Gu et al., 2022)[4] by adding an input-dependent selection mechanism and a hardware-scan algorithm (Gu et al., 2024)[3]. Unlike Transformers, which scale quadratically with sequence length (Vaswani et al., 2017)[7], Mamba achieves linear-time computation by avoiding explicit state expansion and reducing memory bottlenecks. Moreover, Mamba addresses a key limitation of earlier SSMs — poor performance on discrete, information-dense tasks like text — by enabling efficient input selection. These advancements offer a promising balance between model efficiency and effectiveness. Our objective is to evaluate Mamba's accuracy and runtime performance on sentiment classification and compare it to Transformer models to better understand the trade-offs for practical NLP deployment.

1.2 Sentiment140 Dataset

For our project, we used the Sentiment140 dataset[2], which has around 1.6 million tweets labeled for sentiment classification. Each tweet comes with a sentiment label that was originally 0 for negative, 2 for neutral, and 4 for positive. We remapped the labels to $0, 1$, and 2 to make the classification simpler. The tweets in this dataset were collected using the Twitter API by searching for positive or negative emoticons, which makes it one of the earlier examples of large-scale distant supervision for sentiment tasks.

1.3 Current Research

In this project, we explore efficient sequence modeling by evaluating Mamba against structured state-space models (SSMs), Transformers, and LSTMs on a natural language sentiment classification task. Transformers, despite their success, suffer from quadratic scaling with sequence length (Vaswani et al., 2017)[7], while SSMs like S4 (Gu et al., 2022)[4] offer near-linear efficiency through stable, diagonalizable parameterization. However, traditional SSMs underperform on discrete, text-heavy tasks. Building on S4, Mamba (Gu et al., 2024)[3] introduces a selective input mechanism and hardware-aware scan to better model dense data like language, combining Transformer-level flexibility with SSM-level efficiency.

2 Methodology

2.1 Model Architecture

SSMs

To understand Mambas design, we first revisit the foundation of Structured State Space Models (SSMs). SSMs offer a highly efficient alternative to Transformer-based architectures for sequential modeling by achieving linear time complexity $\mathcal{O}(L)$ in sequence length L, compared to the quadratic complexity $\mathcal{O}(L^2)$ of Transformer self-attention layers. This efficiency stems from SSM's core mechanism, where each token interacts only with its immediate past, unlike attention which computes interactions between all token pairs.

Mathematically, SSMs operate based on a set of differential equations:

$$h'(t) = Ah(t) + Bx(t), \quad y(t) = Ch(t)$$

where $h(t)$ is the hidden state, $x(t)$ is the input, and A, B, C are learnable matrices. Discretizing these continuous dynamics for practical computation yields the following recurrence:

$$h_t = \bar{A}h_{t-1} + \bar{B}x_t, \quad y_t = Ch_t$$

The hidden state h_t aggregates contributions from the previous hidden state and current input. Expanding recursively leads to:

$$y_t = C\bar{A}^t\bar{B}x_0 + C\bar{A}^{t-1}\bar{B}x_1 + \cdots + C\bar{B}x_t$$

This formulation can be interpreted as a convolution over the sequence, allowing computation using efficient prefix-sum operations. Thanks to this structure, SSMs achieve constant per-step computation time, regardless of sequence length.

However, traditional SSMs suffer from limited modeling capacity on discrete, information-dense data such as natural language. They treat inputs uniformly, lacking the ability to selectively focus on import tokens — a key advantage of Transformer attention.

Mamba

To address this, Mamba introduces the concept of **Selective State Space Models**. In selective SSMs, the transition matrices Δ, B, C are no longer static, but instead dynamically generated based on each input token. This allows the model to selectively weight information, enhancing its ability to model complex sequences where certain tokens are more informative than others. In our TensorFlow reimplementation, this is achieved by projecting each input token into a higher-dimensional space and using learned linear layers to predict per-token Δ, B, C values (illustrated in Figure 1).

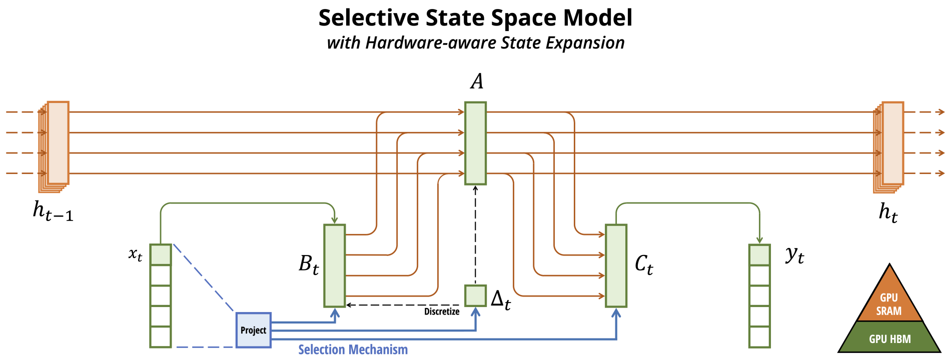

Figure 1: Mamba's selective state space model projects each input token to dynamic parameters (Δ_t, B_t, C_t), discretizes the updates, and performs efficient hidden state transitions using hardware-aware memory placement across GPU SRAM and HBM.[3]

One computational challenge with dynamic selection is that it typically breaks parallelism — if every token's parameters depend on itself, naive sequential computation would be slow. Mamba overcomes this using a **parallel associative scan**. Following prefix-sum operations, Mamba computes cumulative updates efficiently by exploiting the associativity of matrix of matrix operations. It discretizes the dynamic parameters and stores intermediate states entirely in fast SRAM memory on GPUs, avoiding costly memory transfers.

Given an input sequence: $[a_0, a_1, a_2, \ldots, a_{n-1}]$, the prefix-sum produces:

$$[\_, a_0, (a_0 \oplus a_1), (a_0 \oplus a_1 \oplus a_2), \ldots, (a_0 \oplus a_1 \oplus \cdots \oplus a_{n-2})]$$

where $\oplus$ is any associative operation, such as addition or multiplication.

The final Mamba architecture simplifies prior SSM+MLP designs into a compact, efficient block (illustrated in Figure 2):

1. **Linear Upscaling:**
   Each input embedding (e.g., 64 dimensions) is first projected to a higher-dimensional space (e.g., 128 dimensions). This gives the network greater flexibility to manipulate information, making previously inseparable classes more distinguishable.

2. **Local Convolution Mixing:**
   A depthwise 1D convolution is applied independently to each channel. This enables local token interactions, similar to how CNNs capture local patterns.

3. **Selective State Space Update:**
   The convolved sequence is passed through a selective SSM, where each token has its own Δ, B, C parameters generated through learned projections. These dynamics are computed efficiently with parallel scans.

4. **Gated Multiplication:**
   The output of the selective SSM is multiplied by a gate computed from the original input path after a SiLU activation. This gating mechanism acts similarly to attention, allowing the model to modulate how much information from the previous hidden states should flow forward.

5. **Linear Downscaling:**
   Finally, the dimensionality is projected back down to the original embedding size, and a residual connection is added, allowing stable stacking of multiple Mamba blocks.

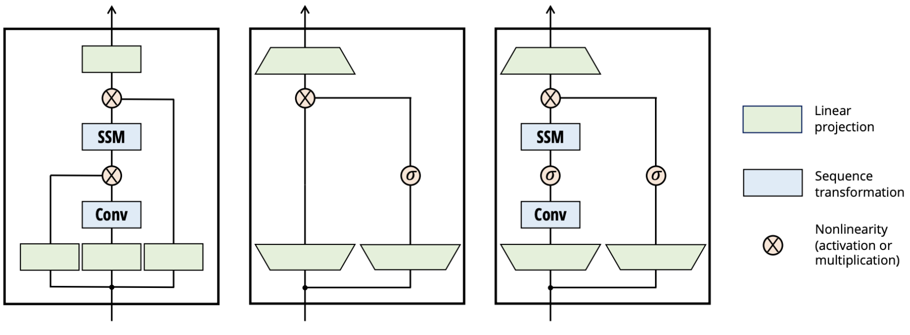

Figure 2: Mamba combines the H3 block with the gated MLP design from modern neural networks into a single unified block. Unlike H3, Mamba replaces the first multiplicative gate with a SiLU activation (Hendrycks and Gimpel, 2016[5]; Ramachandran et al., 2017[6]), and unlike standard MLPs, it adds an SSM module to the main branch, improving both sequence modeling and efficiency.[3]

This sequence of operations — projection, convolution, selective scan, gating, and projection back — defines a single Mamba layer. Stacking multiple such layers, along with layer normalization and residual connections, yields the full Mamba model.

In summary, Mamba combines efficient linear scaling, dynamic input selection, and local convolutional mixing to form a highly effective sequence model. It inherits the efficiency advantages of SSMs while integrating selective mechanisms that allow it to match or exceed the modeling flexibility of Transformers.

2.2 Data Preprocessing

In the preprocessing step, we cleaned the tweets by removing URLs, mentions, hashtags, and special characters, and we also made all the text lowercase to keep it consistent. After cleaning, we used the BERT tokenizer (bert-base-uncased) from Hugging Face to turn the text into token sequences, setting the maximum sequence length to 64 tokens to make sure the inputs were a manageable size. We split the dataset into 98% training, 1% validation, and 1% testing, making sure the label distribution stayed balanced across each split. Finally, we saved the processed data as *CSV* files and loaded them as TensorFlow datasets to use in training our models. This preprocessing setup worked across all the models we trained, including the SSM, LSTM, Transformer, and Mamba models.

2.3 Experiment Design

We re-implemented the Mamba architecture from PyTorch into TensorFlow, faithfully reproducing its selective state space mechanism, convolutional mixing, gating, and hardware-aware scan computations. This translation enabled integration with our TensorFlow training pipeline while maintaining Mamba's structural and runtime properties. To benchmark Mamba, we constructed three comparison models: an LSTM, a Transformer encoder, and a simple Structured State Space Model (SSM). All models were implemented in TensorFlow to ensure consistent evaluation.

For fair comparison, we standardized the model depth, hidden dimensions, and dropout rates across architectures, summarized in Table 1. Each model uses two layers, with hidden sizes matched as closely as possible to control for capacity differences. The Transformer, for example, uses 64-dimensional embeddings and a 128-dimensional feedforward layer, while LSTM uses a 128-dimensional recurrent state.

Model	**Depth**	**Main Layers**	**Hidden Units**	**Dropout**
Mamba	2	HighwayBlock (Selective Scan)	64 internal	0.2
LSTM	2	LSTM	128 units	0.2
Transformer	2	MHA + FFN	64 embed, 128 FF	0.1
SSM	2	SimpleSSMLayer	64	0.1

Table 1: Consistent Model's Structure for Comparison. Mamba uses deeper layers, while LSTM is more prone to overfitting, motivating a slightly higher dropout rate for both models.

2.4 Evaluation Metrics

For this project, we evaluated model performance using two primary metrics: classification accuracy and inference runtime. Accuracy measures the proportion of correctly classified tweets and serves as a standard metric to assess overall model effectiveness in prediction. Inference runtime was used to evaluate each model's efficiency in processing individual samples, which is very important for applications which require real-time or low-latency predictions. These metrics were chosen to balance both the predictive performance and efficiency of the models.

3 Results

3.1 Qualitative Results

We examined a few individual examples to better understand each model's behavior. The following table shows the classification results for several sample tweets, including the true sentiment label and the predictions made by each of the four models.

We observe that for straightforward examples, such as clearly positive or clearly negative tweets, all models generally predicted correctly. However, for more ambiguous or nuanced tweets, slight differences emerged between models. These examples highlight the strengths and weaknesses of each model beyond just aggregate metrics (See Table 2).

Tweet	**Mamba**	**SSM**	**Transformer**	**LSTM**	**True Label**
"relaxing.."	Positive	Positive	Positive	Negative	Positive
"Haha, I agree, they rock and they are awesome."	Negative	Negative	Positive	Negative	Positive
"got cold and just can't be botherd with meself tbh. what happend!?! i hate some peopleon twitter tbh. i'm off to the pub soon"	Negative	Positive	Negative	Negative	Negative
"Nice one bruv! Hope you can make it along"	Positive	Positive	Positive	Negative	Positive

Table 2: Comparison of model predictions across example texts.

3.2 Quantitative Results

We evaluated the performance of our four different models—Mamba, SSM, Transformer, and LSTM—on the sentiment classification task, measuring both accuracy and inference efficiency using inference time. Below we present and analyze the key quantitative findings.

Across the models, Mamba achieved the highest accuracy at 82.32%, followed closely by Transformer (81.86%) and SSM (81.76%). LSTM trailed significantly, achieving only 49.98% accuracy, suggesting it struggled on this task (See Figure 3). Overall, SSM, Transformer, and Mamba performed relatively well in terms of classification accuracy after a single epoch, and LSTM's poor performance could be due to its sequential processing limitations and undertraining.

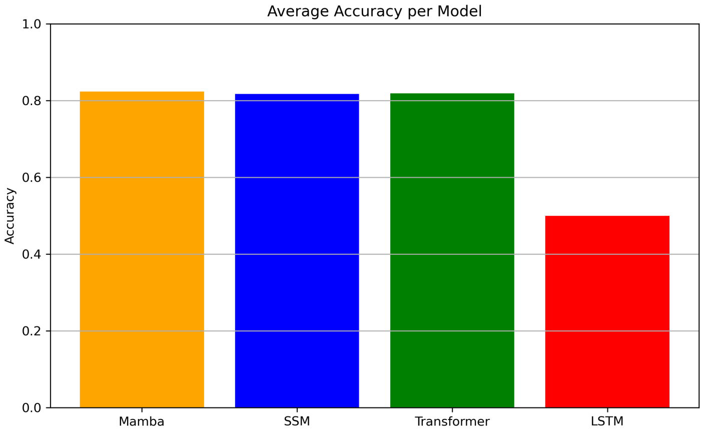

Figure 3: Average accuracy per model

When comparing inference efficiency, SSM performed the fastest, with an average of only 0.0068 seconds per tweet, followed closely by Transformer at 0.00767 seconds. Mamba was moderately fast at 0.02704 seconds, while LSTM was by far the slowest with 0.06191 seconds per sample (See Figure 4). Overall, SSM and Transformer demonstrate highly efficient inference, aligning with expectations for models optimized for sequence processing. Mamba, although slightly slower than SSM and Transformer, still showed great runtime advantages compared to LSTM. LSTM's high runtime highlights the inefficiency of traditional sequential models.

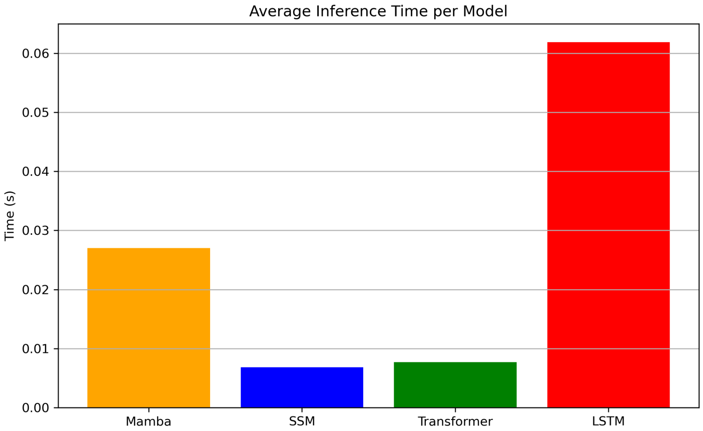

Figure 4: Average inference time per model

Figure 5 illustrates how each model's inference time scales with input sequence length (measured by non-padding tokens). Overall, SSM and Transformer maintain consistently low and stable runtimes across all sequence lengths. Mamba runtime remains stable, but slightly higher than SSM and Transformer. LSTM consistently has the slowest inference, with minimal sensitivity to sequence length but an overall high baseline runtime. No dramatic increases in runtime were observed across the tested range of sequence lengths. Therefore, at the current scale, the models' overall architecture and internal overhead have a bigger impact on inference time than the sequence length.

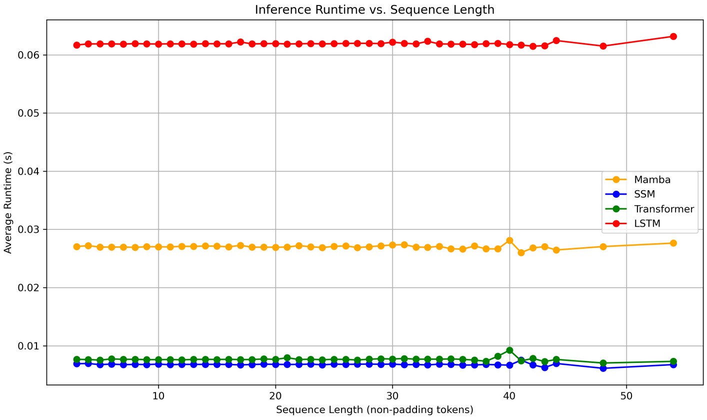

Figure 5: Inference runtime vs. Sequence length per model

4 Challenges

The most challenging part of our project was translating the Mamba model from PyTorch to TensorFlow. In the original PyTorch implementation, Mamba heavily relies on selective scan operations and dynamic state updates, which are efficiently handled using custom CUDA kernels for fast low-level memory access and parallel computation. However, TensorFlow does not directly support these custom operations, and replicating them required careful design.

To address this, we reconstructed each part of the selective scan manually using TensorFlow-native operations, including `tf.einsum` for batched matrix multiplications, `tf.cumsum` for cumulative sums along the sequence dimension, and custom reshaping and slicing logic to align tensor shapes. Our goal was to simulate the original parallel scan behavior efficiently within TensorFlow's computational graph, while preserving both functional correctness and hardware efficiency. During this process, we encountered significant challenges around tensor shape mismatches. Since Mamba's dynamic state updates involve multi-dimensional tensors — typically shaped as (B, L, D, N) — even small misalignments in reshaping, broadcasting, or contraction across dimensions often caused silent errors or inefficient computations. Diagnosing these bugs required thorough inspection of tensor shapes at each computation step, often inserting intermediate assertions and printouts.

Overall, reconstructing Mamba's core mechanisms without access to custom kernels demanded a detailed understanding of both the algorithmic logic and TensorFlow's lower-level tensor manipulation tools. This part of the project was particularly valuable for deepening our understanding of efficient sequence modeling and TensorFlow's capabilities.

5 Reflection

We are pleased with how our project has progressed so far. Our base goals were to successfully re-implement the Mamba architecture in TensorFlow, train it on a tweet sentiment classification task, and benchmark it against strong baselines like LSTM, Transformer, and a simple SSM model. We successfully achieved these goals, and our initial subset testing results showed that Mamba achieved slightly higher accuracy compared to Transformer and SSM, though at the cost of somewhat higher inference time. Overall, the project has validated Mamba's potential for strong predictive performance, even if its runtime efficiency advantages were less pronounced on shorter and simpler sequences like tweets.

Our approach evolved over time as we began to realize the importance of building a unified, standardized evaluation framework across all models. Initially, we focused mainly on getting Mamba functional on synthetic data. As the project matured, we moved toward developing consistent training, testing, and logging pipelines to fairly compare accuracy and runtime across architectures. Each group member actively contributed by troubleshooting unexpected model behavior, brainstorming solutions collaboratively, and deepening their understanding of models through hands-on debugging. This collaborative problem-solving process significantly accelerated our learning and helped us grow more confident in adapting complex architectures. If we had more time, we would conduct hyperparameter tuning, sequence-length scaling experiments, and further ablation studies on Mamba's components (e.g., compare Mamba's performance with and without input-dependent parameters, in order to better understand how important this feature is for the model's effectiveness). In addition, we remain mindful of potential ethical concerns such as model bias and fairness, and future work could explore evaluating the models' predictions across different demographics and input domains to ensure more equitable performance. Our biggest takeaways from this project include the challenges of re-implementing research models across frameworks, the practical nuances of benchmarking model efficiency, and a deeper appreciation for how different models perform in real-world, short-sequence tasks like tweet sentiment analysis.

References

[1] Github repository:. https://github.com/anyfruit/mamba-sentiment-dataset.git.

[2] Sentiment140 dataset. https://www.kaggle.com/datasets/kazanova/sentiment140. Available at: Kaggle.

[3] Albert Gu, Tri Dao, Polina Kirichenko, Xuechen Zhai, Christopher Re, and Abhimanyu Dubey. Mamba: Linear-time sequence modeling with selective state spaces. https://arxiv.org/abs/2312.00752, 2024. arXiv:2312.00752.

[4] Albert Gu, Tri Dao Goel, Shreyas He, Abhishek Rudra, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. https://arxiv.org/abs/2111.00396, 2022. arXiv:2111.00396.

[5] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. https://arxiv.org/abs/1610.02136, 2016. arXiv:1610.02136.

[6] Prajit Ramachandran, Peter Liu, and Quoc V. Le. Diversity and depth in per-example routing models. https://arxiv.org/abs/1711.09130, 2017. arXiv:1711.09130.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. https://arxiv.org/abs/1706.03762, 2017. arXiv:1706.03762.