Mamba Biometric EKG Analysis Technology (MambaBEAT)

Sergio Rodríguez Vidal **Álvaro Pereira Chagoyen** **Antonio Mora Abós**
Jaime Paz Rodríguez
Comillas Pontifical University ICAI
sergio.rodriguez.vidal@sap.com
{pere03,antoniomoraabos,jaimepazri}@alu.comillas.edu

Abstract

Electrocardiography (ECG/EKG) is a common, noninvasive diagnostic tool. The world of EKG investigation has significantly evolved thanks to convolutional neural network models like ResNet, Transformer and automatic interpretation algorithms. As an evolution to these technologies, we propose Mamba Biometric EKG Analysis Technology (MambaBEAT), a new framework built around the Mamba model. By incorporating Selective SSMs and leveraging hardware-aware parallel algorithms, MambaBEAT achieves fast inference and linear scalability in sequence length. This cutting-edge technology has the capacity to transform EKG analysis by utilizing the capabilities of the Mamba model, thereby improving diagnostic precision and advancing patient care. The entire codebase for Mamba Biometric EKG Analysis Technology (MambaBEAT) is open-source and available on our GitHub repository.

1 Introduction

Cardiovascular diseases (CVDs) continue to be a global health concern, affecting more than half a billion people worldwide and **accounting for close to a third of all deaths globally**. In the United States alone, one person dies every 33 seconds from CVD, highlighting the urgent need for effective diagnostic tools.

Electrocardiography (ECG/EKG) has emerged as a noninvasive diagnostic tool that is **increasingly supported by automated interpretation algorithms**. These algorithms leverage deep learning models, such as Transformers (HeartBeit[1]), which have revolutionized the field of sequence modeling across various modalities including language, audio, and genomics. However, these models often suffer from computational inefficiencies, particularly when dealing with long sequences.

Addressing these challenges is **Mamba**[2], a novel sequence modeling architecture that boasts a 5× higher throughput than Transformers and exhibits linear scaling in sequence length. This makes it particularly suited for handling real data up to million-length sequences. Mamba has been shown to **outperform Transformers of the same size** and even match the performance of Transformers twice its size, both in pretraining and downstream evaluation.

Leveraging the advantages of Mamba, we introduce **Mamba Biometric EKG Analysis Technology (MambaBEAT)**, a new framework for EKG analysis. MambaBEAT aims to harness the power of Mamba to drive advancements in EKG classification tasks, pushing the boundaries of what is currently achievable with state-of-the-art models.

In our study, we utilize the **PTB-XL EKG dataset**[3], a large publicly available electrocardiography dataset, to train and evaluate MambaBEAT. This dataset provides a rich source of EKG sequences, allowing us to thoroughly test the capabilities of our model.

Preprint. Under review.

Furthermore, we propose **a comprehensive set of accuracy metrics** to evaluate the performance of our model and several analysis methods to gain deeper insights into the model's behavior. These methods allow us to understand how MambaBEAT makes its predictions, which is crucial for trust and interpretability in medical applications.

2 State Space Models

Traditional state space models, or SSM, are a mathematical framework used to describe the dynamics of a continuous system[4]. They form the foundation of the Mamba model, which is the focus of this paper.

SSMs are defined by a simple equation. This equation takes a 1-D input signal, $x(t)$, and maps it to an N-D latent state, $h(t)$. It then projects this state to a 1-D output signal, $y(t)$.

The equation is as follows:

$$\begin{aligned} h'(t) &= Ah(t) + Bx(t) \\ y(t) &= Ch(t) + Dx(t) \end{aligned} \tag{1}$$

In a deep sequence model, we use the SSM as a black-box representation. The parameters A, B, C, D are learned through gradient descent. For the rest of our discussion, we'll leave out the parameter D in Equation 1. We can do this because the term Dx can be seen as a skip connection, which is easy to compute.

2.1 Discretization

In practical scenarios, continuous measurement or control of systems is not feasible, despite the inherently continuous nature of reality. This renders the handling of continuous derivatives challenging.

The transition from a continuous SSM to a discrete SSM:

$$\begin{aligned} h_t &= \bar{A}h_{t-1} + \bar{B}x_t \\ y_t &= Ch_t \end{aligned} \tag{2}$$

can be achieved through a series of mathematical transformations (e.g. discretization rules), as shown in Appendix B. These transformations lead to the Zero-Order Hold (ZOH) Discretization Rule (Equation 3) and the Bilinear Discretization Rule (Equation 4), which are fundamental to the operation of **Structured state space models (S4)**.

$$\bar{A} = \exp(A\Delta), \quad \bar{B} = (\Delta A)^{-1}(exp(\Delta A) - I)\Delta B \tag{3}$$

$$\bar{A} = (I - \frac{1}{2}\Delta A)^{-1}(I + \frac{1}{2}\Delta A), \quad \bar{B} = (I - \frac{1}{2}\Delta A)^{-1}\Delta B \tag{4}$$

Equation 2 is a **sequence-to-sequence map** instead of function-to-function. In this equation, the state at time t, h_t, depends only on the previous time step, h_{t-1}, and the input at time t, x_t. This property is known as the **Markov property**, which states that the future state depends only on the current state and not on the sequence of events that preceded it. In addition to the Markov property, Equation 2 also exhibits **Linear Time-Invariance (LTI)** (Appendix A).

2.2 Computation

After the discretization of the parameter matrices (Δ, A, B, C) into $(\bar{A}, \bar{B})$, the S4 model can be computed in two primary ways: **linear recurrence** and **convolution**.

1. **Linear Recurrence**: In this method, the computation of the model is performed iteratively, using Equation 2. Linear recurrences can be computed very efficiently with **linear or near-linear scaling in sequence length** $O(L)$. However this is a sequential process, where each step depends on the previous one and is therefore not parallelizable. While this method can be computationally intensive, it is straightforward and easy to implement.

2. **Convolution**: This method is more commonly used because it allows for parallelization in training, which can significantly speed up the computation process. The Convolution method involves the application of a function or a kernel over the input data. This allows Equation 2 to be rewritten as Equation 5:

$$\begin{aligned} \bar{K} &= \{C\bar{B}, C\bar{A}\bar{B}, \dots, C\bar{A}^k\bar{B}, \dots\} \\ y(t) &= \bar{K} * x(t) \end{aligned} \tag{5}$$

where $y(t)$ can be rapidly computed using Fast Fourier Transform thanks to the Convolution theorem. Further details about SSM Computation in Appendix C.

2.3 Long-Range Dependencies with HiPPO

Structured SSMs are so named because computing them efficiently also requires imposing structure on the A matrix. Prior models suggest a special initialization based on the HiPPO theory.

HiPPO is a framework for compressing continuous and discrete data projecting them onto a polynomial basis, which means the polynomial coefficients will be stored in the hidden state[5].

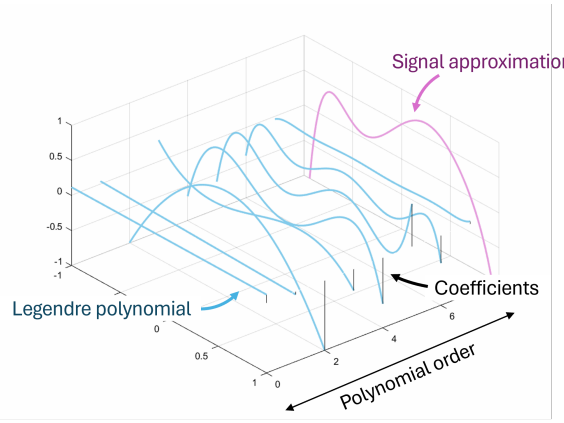

Figure 1: Using Legendre polynomials to approximate a signal.

Most used are **Legendre Polynomials** scaled through a window over time (to avoid forgetting), resulting on the **HiPPO-LegS** matrix[6], defined element-wise as:

$$A_{nk} = \begin{cases} \sqrt{(2n+1)}\sqrt{(2k+1)} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k \end{cases} \tag{6}$$

This matrix has quite interesting properties: invariant to timescale, fast to compute and has **bounded gradients** (avoids the vanishing/exploding gradients problem).

A variety of approximations are made to the HiPPO matrix[7]. In the case of **S4D-Real**, all elements are computed in the same manner as HiPPO's diagonal.

3 Selective State Space Models

While S4 Models have shown to be more computationally efficient and capable of handling long-range dependencies better than some state-of-the-art models like the Transformer, they do have their limitations.

One of the key challenges with S4 Models is their performance in context-dependent tasks. This is partly due to the Linear Time Invariance (LTI) property of S4 Models, which makes the input-dependence seen in the attention mechanism[8] of Transformers impossible. The attention mechanism in Transformers allows the model to focus on different parts of the input sequence when producing an

output, thereby capturing the context more effectively. This level of context sensitivity is something that S4 Models, due to their LTI property, struggle to achieve.

To address this limitation, Albert Gu and Tri Dao introduced **Selective State Space Models (SSMs)** in their Mamba model[2]. Selective SSMs allow the parameter matrices (Δ, B, C) be input dependent. Specifically, for an input $x_t : (B, L, D)$, they define:

- $B_t : (B, L, N) \leftarrow \text{Linear}_N(x)$
- $C_t : (B, L, N) \leftarrow \text{Linear}_N(x)$
- $\Delta_t : (B, L, D) \leftarrow \text{Broadcast}_D(\text{Linear}_N(x))$.

This mechanism is inspired by concepts such as gating mechanisms (in LSTMs[9] and GRUs[10]) and linear attention[11]. The resulting equation of the selective SSMs is therefore:

$$\begin{aligned} h_t &= \bar{A}_t h_{t-1} + \bar{B}_t x_t \\ y_t &= C_t h_t \end{aligned} \tag{7}$$

where $\bar{A}_t$ and $\bar{B}_t$ are the "discretized" matrices A and B_t with respect to Δ_t.

3.1 Parallelization

With the introduction of dynamic parameters in Mamba, the **precomputation of the convolution kernel becomes unfeasible**. However, to circumvent this challenge and enable parallelization, Mamba incorporates three classical techniques: **parallel scan, kernel fusion, and recomputation**.

A simple and common parallel algorithm building block is the all-prefix-sums operation. This operation on an array of data is commonly known as **scan**.

Given an array of elements x and a binary associative operator $\oplus$, the scan operation computes a new array y where each element is the sum of all preceding elements in the original array.

$$\begin{cases} y_i = y_{i-1} \oplus x_i & \forall i \quad 1 \leq i < n \\ y_i = x_i & i = 0 \end{cases} \tag{8}$$

The solution to this problem seems inherently sequential, but for which there is an efficient parallel algorithm which allows the prefix sums to be computed faster on a GPU. One of the best-known parallel scans when $\oplus$ is addition is the **Blelloch Scan**[12].

In the case of Selective SSMs, the recurrent formula of the SSM model can also be thought of as a scan operation, in which each state is a weighted sum of the previous states and the current input.

Mamba's **selective scan** is applied to initial elements $e_{0:N}$ defined as:

$$e_t = (e_{t,a}, e_{t,b}) := (\bar{A}_k, \bar{B}_t x_t) \tag{9}$$

where $\bar{A}_t$, $\bar{B}_t$ and x_t are defined in Equation 7. Mamba's **parallel operator** is then defined as:

$$e_i \oplus e_j = (e_{j,a} e_{i,a}; e_{j,a} e_{i,b} + e_{j,b}) \tag{10}$$

It is almost trivial to prove that this operator is associative (Proof D.2). The Blelloch scan can be transformed to the selective scan by having it use this scan operator instead of traditional addition (Appendix D).

One of the challenges with parallelization is the **limited data transfer speed** of recent GPUs between their small yet highly efficient SRAM and their large but slightly less efficient HBM. This constant back-and-forth of data between SRAM and HBM (DRAM) can slow things down. Mamba, like Flash Attention, attempts to limit the number of trips between HBM and SRAM.

To make a sequence of operations faster, Mamba fuses CUDA kernels (**kernel fusion**) to produce a custom CUDA kernel that performs all the operations involved in the Selective SSM one after another without copying the intermediate results to the HBM and just copy the final results.

Finally, Mamba avoids saving the intermediate states, which are necessary for backpropagation. Through **recomputation**, Mamba recomposes the intermediate states of the model during the backward pass. Although this might seem inefficient, it is proven that this action is less costly than reading all the intermediate states from the DRAM.

4 The Mamba Block

Mamba is a type of architecture that combines two key components: The **H3 block** (which is the basis for the most well-known SSM architectures) and the **MLP block** (a common element in modern neural networks).

The Mamba architecture begins by linearly projecting input embeddings for enhanced data representation. These expanded embeddings are bifurcated into a main pathway and a residual connection.

The main pathway applies a 1-Dimensional convolution to the expanded inputs, preserving sequence information by capturing local dependencies. The convolution output undergoes a SiLU (Swish) activation, introducing non-linearity and aiding complex representation learning.

This transformed data is then processed by the Selective SSM, a flexible sequence transformation module. Concurrently, the residual connection also undergoes a SiLU activation.

The outputs from both pathways are element-wise multiplied, allowing retention of original information potentially lost in the main pathway transformations. Finally, the input embeddings are reverted to their original size.

Mamba stacks multiple of these Mamba blocks, interleaved with standard normalization, and uses their output as the input for the next Mamba block.

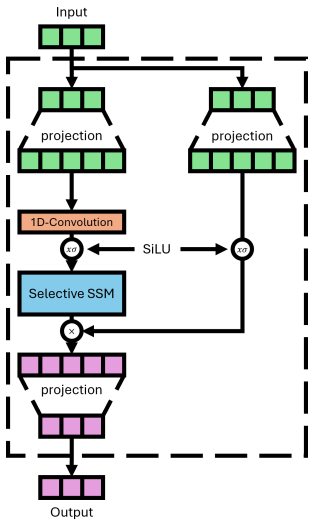

Figure 2: Overview of the Mamba Block.

5 Our proposal: MambaBEAT

Our proposal, **MambaBEAT**, is a novel framework for EKG analysis that leverages the power of the Mamba model. This framework is designed to improve diagnostic precision and advance patient care by utilizing the capabilities of the Mamba model.

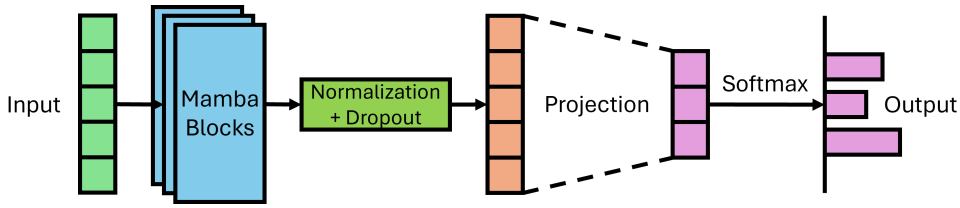

Figure 3: MambaBEAT model.

The MambaBEAT model is implemented in PyTorch and consists of several key components:

- **Mamba Blocks**: These are the core building blocks of the MambaBEAT model. Each Mamba Block is a combination of an H3 block and an MLP block, which are common elements in modern neural networks. The Mamba Blocks are responsible for linearly projecting input embeddings for enhanced data representation and applying a 1-Dimensional convolution to the expanded inputs to preserve sequence information.
- **Normalization and Dropout**: After the data is processed through the Mamba Blocks, it undergoes normalization (**RMSNorm**) and dropout. This step helps to prevent overfitting and ensures that the model generalizes well to unseen data.
- **Projection**: The normalized and dropout-processed data is then transformed through a linear layer. This step is crucial for transforming the data into a space where the final classification can be performed.

- **Softmax Activation**: The final step in the MambaBEAT model is the application of a softmax activation function. This function outputs a probability distribution over the target classes, which allows for the interpretation of the model's output as class probabilities.

The MambaBEAT model is designed to be **flexible and scalable**, with parameters such as the number of layers, the dimension of the latent state, the expansion factor, the rank of the discrete-time system, the kernel size, and the dropout rate all being configurable. This makes MambaBEAT adaptable to a wide range of EKG analysis tasks.

6 Benchmarking

In this section we will compare the performance of MambaBEAT with other state-of-the-art models in EKG analysis.

6.1 Dataset

The PTB-XL dataset[3] encompasses 21,837 clinical **12-lead EKG records** collected from 18,885 patients, with a gender distribution of 52 males and 48 females. Each EKG has a **length of 10 seconds**, recorded in two different resolutions. The EKG records are annotated according to the SCP-EKG standard and categorized into three non-mutually exclusive groups: diagnostic (diag.), form, and rhythm. These annotations comprise a total of 71 statements, consisting of 44 diagnostic, 12 rhythm, and 19 form statements. Notably, four form statements overlap with the diagnostic ECG statements.

The diagnostic statements are further organized into **five coarse superclasses**: NORM (normal EKG), CD (conduction disturbance), MI (myocardial infarction), HYP (hypertrophy), and STTC (ST/T changes), with 24 sub-classes. For comprehensive insights into the dataset and its annotation scheme, we refer interested readers to the original publication.

Figure 4 shows an EKG sample obtained from the PTB-XL dataset and plotted using our "`plot_ekg`" function:

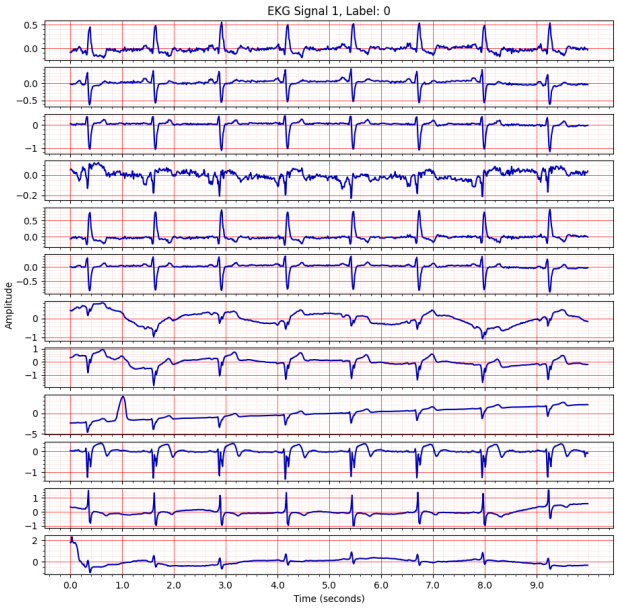

Figure 4: Normal EKG sample from the PTB-XL dataset.

6.2 Multivariate EKG Classification Algorithms

In this subsection, we discuss the multivariate classification algorithms that we have targeted for benchmarking. These algorithms are chosen due to their ability to **handle multiple output variables**, which is a common scenario in EKG analysis where we need to predict multiple diagnostic statements.

- **Convolutional Neural Networks (CNNs)**: CNNs are particularly effective for EKG analysis due to their ability to automatically learn and extract robust features from raw EKG signals. They are designed to **process 1-D signals**, making them suitable for EKG data. We used both standard CNNs and ResNet architectures for our benchmarking. The ResNet architectures included ResNet50, ResNet101, and ResNet152. These models, with their deep layers and skip connections, are capable of capturing complex patterns in the EKG data.
- **Recurrent Neural Networks (RNNs)**: RNNs, including standard RNNs, Long Short-Term Memory (LSTM), and Bidirectional LSTM (Bi-LSTM), are used due to their ability to handle sequential data effectively. This is crucial for EKG analysis as EKG data is time-series data where the order of data points matters. LSTM and Bi-LSTM, with their gating mechanisms, are particularly **effective at capturing long-term dependencies** in the data, which can be important for EKG analysis.
- **Other**: We also used a **feature model** as a baseline classifier. This model uses features such as patient's age, gender, etc., for classification.

For reasons of clarity, we **only report the performance for selected representatives** including the best-performing method for each group.

6.3 Accuracy metrics

In the context of EKG multi-label classification, where each EKG can have a single or multiple diagnoses (including contradictory ones), we have adopted two distinct approaches.

The first approach is a **binary classification** for each label. This method checks if the diagnosis for each label would be "reasonable". The labels are encoded using **One Hot Encoding**, which transforms categorical data into a format that can be provided to machine learning algorithms to improve prediction. In the context of EKG data, this means that each diagnosis (NORM, CD, MI, HYP, STTC) is considered as a separate binary label. The model then predicts whether each diagnosis is present (1) or absent (0). This approach allows the model to predict multiple diagnoses for a single EKG, reflecting the complex and multifaceted nature of cardiac conditions. The loss function used in this approach is the **Binary Cross-Entropy (BCE) loss**.

The second approach is a **multi-class classification**, where the model predicts a single output class. This class is the argmax logit of the softmax values, essentially predicting the most "reasonable" diagnosis. The labels are encoded using a **Label Encoder**, which converts each class under specified feature to a numerical value. This approach simplifies the problem by forcing the model to choose the most likely diagnosis. It can be particularly useful in scenarios where a clear and decisive diagnosis is required. Importantly, this approach also provides the probabilities for each of the possible diagnoses, enhancing the explainability of the model. The loss function used in this approach is the **Cross-Entropy loss**.

Both approaches have their merits. The binary classification approach is more flexible and can potentially provide **a more comprehensive view of the patient's cardiac health**. However, it may also lead to predictions that are difficult to interpret, especially when **contradictory diagnoses** are predicted.

On the other hand, the multi-class classification approach provides a clear and unambiguous prediction, but it may oversimplify the problem and miss important nuances in the EKG data. However, the ability to provide probabilities for each diagnosis **enhances its interpretability**.

6.4 Results

The **benchmarking results** presented in Tables 1 and 2 demonstrate the performance of various models in binary and multi-class classification tasks, respectively.

Models	Training		Validation		Test
	Accuracy	Loss	Accuracy	Loss	Accuracy
CNN	**0.88**	**0.26**	0.77	0.75	0.758
RestNet101	0.745	0.54	0.74	0.587	-
BiLSTM	0.82	0.35	0.84	0.36	0.8367
MambaBEAT	0.86	0.32	**0.845**	**0.35**	**0.845**

Table 1: Performance in binary classification on training, validation, and test sets

Models	Training		Validation		Test
	Accuracy	Loss	Accuracy	Loss	Accuracy
CNN	-	-	-	-	-
RestNet152	0.468	1.417	0.4	1.488	0.3857
BiLSTM	0.58	**1.05**	0.56	**1.08**	0.557
MambaBEAT	**0.64**	1.26	**0.59**	1.306	**0.579**

Table 2: Performance in multi-class classification on training, validation, and test sets

As can be observed, the **multi-class classification models generally exhibit lower accuracies** compared to their binary classification counterparts. This is a logical outcome considering the nature of multi-class classification, where the model is tasked with predicting only one label from multiple possibilities. Other potential diagnoses are not considered in this scenario, which can augment the error rate. Despite this, as mentioned earlier, multi-class classification provides valuable insights as it forces the model to make a **decisive prediction**, which can be particularly useful in certain clinical scenarios.

Among the models benchmarked, **MambaBEAT consistently outperforms the others in both binary and multi-class classification tasks**, demonstrating its robustness and versatility. It is noteworthy that the performance of MambaBEAT is closely matched by the BiLSTM model, indicating the effectiveness of recurrent neural networks in EKG classification tasks.

In terms of computational efficiency, each epoch of **MambaBEAT took approximately 49 seconds on a 4090 NVIDIA GPU**, which is about 10% slower than BiLSTM. However, the saved **MambaBEAT model is significantly smaller**, being approximately five times smaller than the BiLSTM model. This makes MambaBEAT a more storage-efficient choice.

All models were trained using the **AdamW optimizer and StepLR learning rate scheduler**. The models were **tuned differently to optimize each loss function** (Binary Cross-Entropy for binary classification and Cross-Entropy for multi-class classification). Except for the second BiLSTM (multi-class classification) with a batch size of 64 and the second MambaBEAT with a batch size of 256, all other models were trained with a **batch size of 128**. MambaBEAT was trained using default parameters. All models achieved these accuracies **under 200 epochs**.

7 Conclusion

In summary, our newly developed MambaBEAT model has demonstrated **promising results** in both binary and multi-class EKG classification tasks, outperforming other state-of-the-art models. As the field continues to evolve, **we anticipate seeing more sophisticated Mamba models**, with innovative approaches and comprehensive solutions to handle the complexity and diversity of EKG data. We encourage researchers and practitioners to **further optimize MambaBEAT** to achieve even higher accuracies.

One potential avenue for **enhancing the performance of MambaBEAT is through pretraining**. Pretraining the model on a large and diverse dataset could allow it to learn general features and patterns in EKG data, which could then be fine-tuned on a specific task. This approach has the potential to significantly boost the performance of MambaBEAT, making it an even more powerful tool for EKG analysis. **We look forward to seeing the advancements in this exciting area of research**.

References

[1] HeartBEiT: Vision Transformer for Electrocardiogram Data Improves Diagnostic Performance at Low Sample Sizes. [Online]. Available: https://arxiv.org/abs/2212.14040

[2] Mamba: Linear-Time Sequence Modeling with Selective State Spaces. [Online]. Available: https://arxiv.org/abs/2312.00752

[3] PTB-XL, a large publicly available electrocardiography dataset. [Online]. Available: https://doi.org/10.13026/x4td-x982

[4] Efficiently Modeling Long Sequences with Structured State Spaces. [Online]. Available: https://arxiv.org/pdf/2111.00396.pdf

[5] HiPPO: Recurrent Memory with Optimal Polynomial Projections. [Online]. Available: https://arxiv.org/abs/2008.07669

[6] How to Train Your HiPPO: State Space Models with Generalized Orthogonal Basis Projections. [Online]. Available: https://arxiv.org/abs/2206.12037

[7] On the Parameterization and Initialization of Diagonal State Space Models. [Online]. Available: https://arxiv.org/pdf/2206.11893.pdf

[8] Attention Is All You Need. [Online]. Available: https://arxiv.org/abs/1706.03762

[9] Long Short-Term Memory [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[10] Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. [Online]. Available: https://arxiv.org/pdf/1412.3555.pdf

[11] Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. [Online]. Available: https://arxiv.org/abs/2006.16236

[12] Prefix sums and their applications. [Online]. Available: https://api.semanticscholar.org/CorpusID:60459178

[13] Deep Learning for ECG Analysis: Benchmarks and Insights from PTB-XL. [Online]. Available: https://arxiv.org/pdf/2004.13701.pdf

[14] Combining Recurrent, Convolutional, and Continuous-time Models with Linear State-Space Layers. [Online]. Available: https://arxiv.org/abs/2110.13985

[15] MambaByte: Token-free Selective State Space Model. [Online]. Available: https://arxiv.org/abs/2401.13660

[16] Vision Mamba: Efficient Visual Representation Learning with Bidirectional State Space Model. [Online]. Available: https://arxiv.org/abs/2401.09417

[17] Neural Machine Translation by Jointly Learning to Align and Translate. [Online]. Available: https://arxiv.org/abs/1409.0473

[18] GPU Gems 3 by Hubert Nguyen. [Online]. Available: https://developer.nvidia.com/gpugems/gpugems3

A Linear Time Invariance

A key characteristic of equations 1, 2 and 5 is the unchanging nature of the model's dynamics over time. This means that the parameters (Δ, A, B, C), and by extension (A, B), remain **constant across all time-steps**. This characteristic is known as **time invariance**. The system is also **linear**, which means that the response to a sum of inputs is equal to the sum of the responses to each input individually. This is a very useful property because it allows for powerful mathematical techniques like **superposition and convolution** to be used in the analysis of the system's behavior.

B Discretization Rules

To transition **from a continuous to a discrete model**, the derivative in the State Equation 1 must be eliminated. This can be achieved by integrating both sides of the equation. However, before integration, both sides of the equation are multiplied by an exponential matrix, as shown in Equation 11:

$$e^{-At}h'(t) = e^{-At}Ah(t) + e^{-At}Bx(t) \tag{11}$$

This manipulation is a clever application of the **product rule** from calculus, where $u = h(t)$ and $v = e - At$. Applying this rule backwards results in Equation 12:

$$\frac{d}{dt}[e^{-At}h(t)] = e^{-At}Bx(t) \tag{12}$$

Now, both sides of the equation **can be integrated more easily**, as shown in Equation 13:

$$\begin{aligned} \int_0^t \frac{d}{dt}[e^{-A\tau}h(\tau)]d\tau &= \int_0^t e^{-A\tau}Bx(\tau)d\tau \\ e^{-At}h(t) - h(0) &= \int_0^t e^{-A\tau}Bx(\tau)d\tau \end{aligned} \tag{13}$$

Multiplying by e^{At} and moving the constant term $h(0)$ to the right side leads to the following equation:

$$h(t) = e^{At}h(0) + \int_0^t e^{A(t-\tau)}Bx(\tau)d\tau \tag{14}$$

The next step is to define the discrete state $h_k = h(\Delta k)$ and plug it into the analytical solution, which results in Equation 15:

$$h_k = e^{Ak\Delta}h(0) + \int_0^{k\Delta} e^{A(k\Delta-\tau)}Bx(\tau)d\tau \tag{15}$$

It is now easy to **predict the next state based on the previous one**, as shown in Equation 16:

$$h_{k+1} = e^{A\Delta}h_k + \int_{k\Delta}^{(k+1)\Delta} e^{A((k+1)\Delta-\tau)}Bx(\tau)d\tau \tag{16}$$

The only remaining task now is to **solve the integral**. Let $u(t) = (k+1)\Delta - \tau$ and $dv = -d\tau$:

$$\begin{aligned} h_{k+1} &= e^{A\Delta}h_k + (\int_0^{\Delta} e^{Av}dv)Bx_k \\ h_{k+1} &= e^{A\Delta}h_k + A^{-1}(e^{A\Delta} - I)Bx_k \end{aligned} \tag{17}$$

B.1 Zero-Order Hold (ZOH) Discretization Rule

From Equation 17, we obtained the Zero-Order Hold (ZOH) Discretization Rule:

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

where $\bar{A} = \exp(A\Delta)$ and $\bar{B} = (\Delta A)^{-1}(exp(\Delta A) - I)\Delta B$ (Equation 3).

B.2 Bilinear Discretization Rule

To obtain the Bilinear Discretization Rule (Equation 4), the Forward Euler Approximation $e^{A\Delta} \approx I + A\Delta$ (First order Taylor Expansion) and Backward Euler Approximation $e^{A\Delta} \approx (I - A\Delta)^{-1}$ are used.

$$\bar{A} = \exp(A\Delta) = (\exp(A\Delta))^{\alpha}(\exp(A\Delta))^{1-\alpha} = (I - \alpha A\Delta)^{-1}(I + (1-\alpha)A\Delta) \tag{18}$$

This family of α-dependent discretization methods on a parameter is called the **Generalized Bilinear Transform (GBT)**. If $\alpha = 0$, then it becomes the **classic Euler method** which is simply the first-order approximation $x(t + \Delta t) = x(t) + \Delta t \cdot x(t)$. If $\alpha = 1$ then it is called the **backward Euler**.

When $\alpha = 0.5$ then the **Bilinear Discretization Rule** is obtained (which preserves the stability of the system):

$$\bar{A} = (I - \frac{1}{2}\Delta A)^{-1}(I + \frac{1}{2}\Delta A), \quad \bar{B} = (I - \frac{1}{2}\Delta A)^{-1}\Delta B \tag{4}$$

C Recurrent and Convolutional Representation of SSMs

As mentioned in Section 2.2, discrete-time SSMs can be computed using two different representations:

As a linear recurrence: The recurrent state $h^{t-1} \in R^{HxN}$ carries the context of all inputs before time t. The current state h_t and output y_t can be computed by simply following Equation 2. Thus, State Space Models are recurrent models with efficient and stateful inference, which can consume a (potentially unbounded) sequence of inputs while requiring fixed computation/storage per time step.

The image below shows the recurrent representation of a State Space Model:

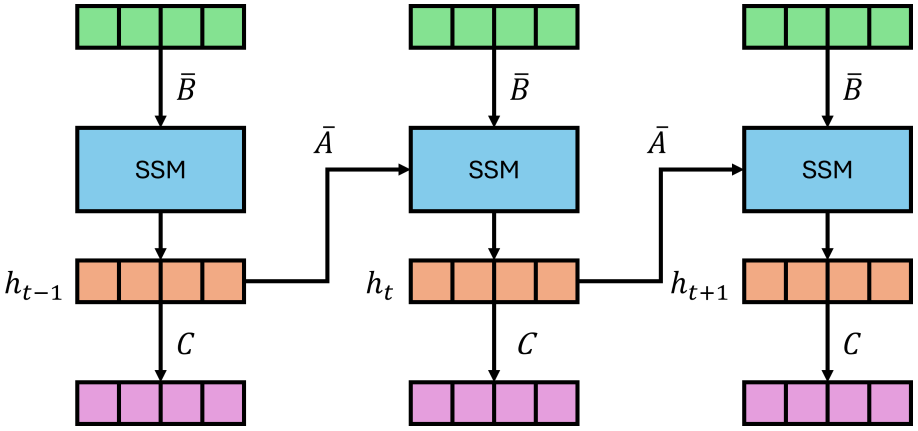

Figure 5: Recurrent representation of SSMs.

The main drawback of this computation method is that it cannot be parallelized using GPUs, resulting in slow training. Hence why, the next representation has become more common.

As a convolution: Let the initial state be $x_{-1} = 0$. Then Equation 2 explicitly yields:

$$y_k = C(\bar{A})^k\bar{B}x_0 + C(\bar{A})^{k-1}\bar{B}x_1 + ... + C\bar{A}\bar{B}x_{k-1} + \bar{B}x_k \tag{19}$$

This expression can be rewritten as Equation 5:

$$\begin{aligned} \bar{K} &= \{C\bar{B}, C\bar{A}\bar{B}, \ldots, C\bar{A}^k\bar{B}, \ldots\} \\ y(t) &= \bar{K} * x(t) \end{aligned} \tag{5}$$

where the kernel K can be precomputed using the Fast Fourier Tranform (FFT) via Theorem 1.

Theorem 1 (Convolution theorem)**.** *Let $x[n]$ and $h[n]$ be sequences of length N, and let $X[k]$ and $H[k]$ be their respective Discrete Fourier Transforms. Then the circular convolution of $x[n]$ and $h[n]$ can be computed by taking the inverse Discrete Fourier Transform of the product of $X[k]$ and $H[k]$.*

Mathematically, this can be represented as:

$$y[n] = \mathcal{F}^{-1}\{X[k] \cdot H[k]\}$$

where: - $y[n]$ is the result of the circular convolution of $x[n]$ and $h[n]$, - $\mathcal{F}^{-1}\{\}$ denotes the inverse Discrete Fourier Transform, - $X[k]$ and $H[k]$ are the Discrete Fourier Transforms of $x[n]$ and $h[n]$ respectively, - $\cdot$ denotes multiplication.

This theorem is the basis for efficient convolution computations using FFT, often referred to as Fast Convolution.

D Selective scan

With the introduction of dynamic parameters in Mamba, the **precomputation of the convolution kernel becomes unfeasible**. However, to circumvent this challenge and enable parallelization, Mamba incorporates a work-efficient parallel scan based on the Blelloch scan.

D.1 Blelloch scan

The Blelloch scan algorithm is predicated on the construction of a **balanced binary tree** from the input data. This tree is then traversed towards (Up-Sweep) and away from (Down-Sweep) the root in order to compute the prefix sum. The **parallelization** arises from the fact that operations at each depth of the binary tree can be performed concurrently, enabling efficient computation of prefix sums.

During the **Up-Sweep phase**, the algorithm performs a bottom-up traversal of the tree, calculating partial sums at each internal node. Upon completion of this phase, the root node (the final node in the array) contains the aggregate sum of all nodes. This stage is also referred to as the reduction phase.

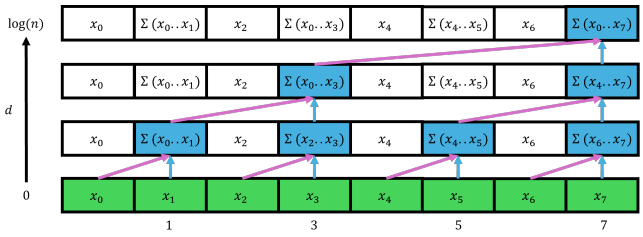

Figure 6: Up-Sweep phase of the Blelloch scan.

The process essentially involves the construction of a binary sum tree. In essence, pairs of numbers separated by a specific step size are added together. This step size is computed as 2^{d+1}, where d represents the current depth of the binary tree.

During the **Down-Sweep phase**, the algorithm traverses the tree in a top-down manner, starting from the root. The partial sums computed during the reduce phase are utilized as the root. The **final prefix sum** is computed independently. The process commences with the insertion of a zero at the root of the tree.

At each node at the current level, the Down-Sweep operation involves two elements. The left element is copied into the space of the right element, following which a binary operation is applied to replace the left element. This sequence is repeated throughout the traversal, ensuring the computation of the prefix sum.

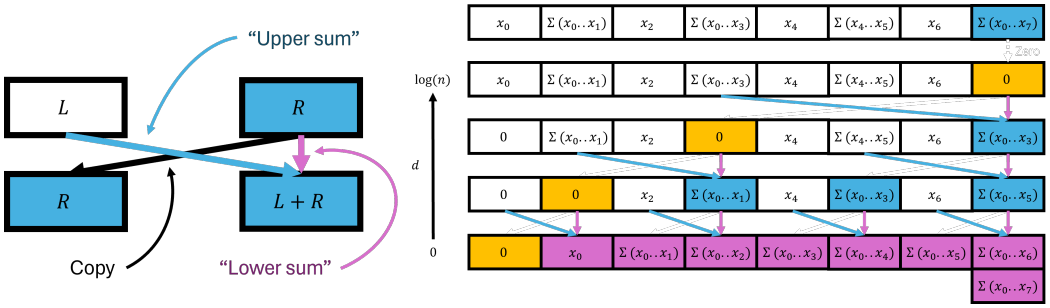

Figure 7: Down-Sweep phase of the Blelloch scan.

Here is a simple **pseudocode** of the Blelloch scan. Notice that the inner loops are processed in parallel for faster computation.

```
procedure blellochScan(A[0..N))
    // Store the original size and last element
    originalSize := N
    lastElement := A[N-1]

    // Calculate the next power of 2
    nextPowerOf2 := 2^ceil(log2(N))

    // Extend the array to the next power of 2 with zeros
    for i := N to nextPowerOf2-1 do
        A[i] := 0
    end for
    N := nextPowerOf2

    // Upsweep phase
    for d := 0 to log(N)+1 do
        step := 2^(d+1)
        for i := step-1 to N-1 by step do in parallel
            A[i] := A[i] + A[i - step/2]
        end parallel for
    end for

    A[N-1] := 0  // clear the last element

    // Downsweep phase
    for d := log(N)+1 down to 0 do
        step := 2^(d+1)
        for i := step-1 to N-1 by step do in parallel
            t := A[i - step/2]
            A[i - step/2] := A[i]
            A[i] := t + A[i]
        end parallel for
    end for

    // Remove the first zero and add the last element
    A := A[1: originalSize]
    A[originalSize-1] := A[originalSize-1] + lastElement
end procedure
```

Figure 8: Pseudocode for Blelloch scan.

D.2 Associative Parallel Scan

Parallel scans take advantage of the fact that the composition of **associative operations** can be computed in any order. We can define binary operations as complex as we want as long as our scan operator is still associative

In Section 3 we discussed how the Blelloch scan can be adapted to compute Selective SSMs using the following scan operator (instead of traditional addition):

$$e_i \oplus e_j = (e_{j,a}e_{i,a}; e_{j,a}e_{i,b} + e_{j,b}) \quad e_t = (e_{t,a}, e_{t,b}) := (\bar{A}_k, \bar{B}_t x_t) \tag{10}$$

Proof. We will **prove the associativity of the operator** $\oplus$ defined in Equation 10, which is used to compute Selective SSMs:

$$\begin{aligned}
(e_0 \oplus e_1) \oplus e_2 &= (\bar{A}_1\bar{A}_0, \bar{A}_1\bar{B}_0x_0 + \bar{B}_1x_1) \oplus e_2 \\
&= (\bar{A}_2(\bar{A}_1\bar{A}_0), \bar{A}_2(\bar{A}_1\bar{B}_0x_0 + \bar{B}_1x_1) + \bar{B}_2x_2 \\
&= ((\bar{A}_2\bar{A}_1)\bar{A}_0, \bar{A}_2\bar{A}_1(\bar{B}_0x_0) + \bar{A}_2\bar{B}_1x_1 + \bar{B}_2x_2) \\
&= e_o \oplus (\bar{A}_2\bar{A}_1, \bar{A}_2\bar{B}_1x_1 + \bar{B}_2x_2) \\
&= e_0 \oplus (e_1 \oplus e_2)
\end{aligned} \tag{20}$$

□

D.3 Pseudocode

Here is a simple pseudocode of the **Selective scan** that uses the operator defined at Equation 10. Notice that the inner loops are processed in parallel for faster computation.

```
procedure selectiveScan(A[0..N), X[0..N])
    // Store the original size and last elements
    originalSize := N
    lastElementA := A[N-1]
    lastElementX := X[N-1]

    // Calculate the next power of 2
    nextPowerOf2 := 2^ceil(log2(N))

    // Extend the arrays to the next power of 2 with zeros
    for i := N to nextPowerOf2-1 do
        A[i] := 0
        X[i] := 0
    end for
    N := nextPowerOf2

    // Upsweep phase
    for d := 0 to log(N)+1 do
        step := 2^(d+1)
        for i := step-1 to N-1 by step do in parallel
            X[i] := A[i] * X[i - step/2] + X[i]
            A[i] := A[i] * A[i - step/2]
        end parallel for
    end for

    X[N-1] := 0  // clear the last element

    // Downsweep phase
    for d := log(N)+1 down to 0 do
        step := 2^(d+1)
        for i := step-1 to N-1 by step do in parallel
            t := X[i - step/2]
            X[i - step/2] := X[i]
            X[i] := A[i - step/2] * X[i] + t
        end parallel for
    end for

    // Remove the first zero and add the last element
    X := X[1: originalSize]
    X[originalSize-1] := lastElementA * X[originalSize-1] + lastElementX
end procedure
```

Figure 9: Pseudocode for Blelloch scan.

The forward and backward methods of this scan algorithm are implemented in Python in our GitHub repository.