Transformer Mamba with Spiking

Tong Miao
tmi@ucdavis.edu
University of California, Davis
Davis, California, USA

1 ABSTRACT

In recent years, there has been developments in spiking neural networks (SNN) which are advantaged over ANNs in many different ways, such as energy efficiency and reduced computational complexity due to the discrete nature of spikes. In the vision space, after the limitations of spiking ResNet models become apparent in higher complexity tasks, recent research has all been focusing on the spiking transformer. Specifically some formulation of the spiking self attention system that is the core of the transformer. In this work, I take a different approach and attempt to introduce Mamba, the relatively novel selective state space model to the SNN space. The primary contribution of this work is the creation of the spiking mamba block and it's relevant spiking S6 selection module. With the spiking mamba block, I test how various hyper parameters affect it's performance, as well as combine it with transformer blocks to see if it can be better than either individually. For results I achieved 89% accuracy on the CIFAR-10 dataset, which while is not state of the art, does signify the potential utility of spiking mamba for future works.

2 INTRODUCTION

Spiking neural networks (SNN) are often hailed as the next generation [13] of neural networks due to its biological similarity, low power requirements [14], and ability to process neuromorphic datasets that are more relevant to how humans view the world. However, it has historically suffered compared to conventional artificial neural networks (ANN) in terms of performance.

Many attempts has been made to correct this, primarily by taking the advancements made in the traditional ANN space into the SNN space. These attempts include the Spiking Resnet [8] architecture to understand residual learning in SNNs, as well as multiple formulations for introducing transformers [18, 25] to the SNN space, all with varying degrees of success as well as faithfulness to the original transformer and self attention architectures [22, 23].

However, all these attention based formulations have a large problem that is exacerbated by SNN models. Namely the computational costs [4, 10] in token processing. The vanilla self attention mechanism as well as it's derivatives relies on generating a value for each token based on every other token that came before it, leading great performance especially in understanding long range dependencies[26] and deeper relations between tokens, but comes at the cost of large amounts of matrix multiplication for giant matrices.

With SNN specifically, both training and inference are not as simple as in ANNs. Due to the stateful and time dependent nature of spike generation, SNNs require multiple time steps [1] per forward pass. These timesteps could range from a lower amount of 4 as used in the SpikeFormer[25], all the way to the 250 steps used in the converted ANN2SNN RestNet-34 [24]. While attention is computationally heavy on it's own, when combined with the many timesteps needed for spiking neural networks, this creates multiple times the computation needed depending on the time steps used. For the backward error propagation, SNNs also require the usage of synthetic gradients [2] to train due to the non differentiable nature of spikes. Due to the multiple timestpes, the training process also require taking the gradients of each time step. All together, this makes attention based mechanisms, which are already expansive, much more so for SNNs.

On the other hand, recent advancements in the ANN space, namely mamba [6], recognized the problem with the heavy costs of transformers and attempted to solve it by combining state space models with a selection mechanism. This work has been largely successful in solving the computational cost issue of transformers and attention in ANNs without compromising much in the quality of the output.

While certain works have criticized [9] mamba and experimentally shown that it cannot compete with transformer in all areas, due to several weaknesses especially related to information retrieval and copying, mamba is still good enough for most tasks, especially where computational power and smaller model sizes are prioritized over achieving the absolute state of the art performance [19]. To this end, the mamba architecture is strongly relevant to the progress of SNNs, which have the computational efficiency issue due to the nature of statefulness over multiple time steps, and it's goal of achieving more efficient power usage compared to ANNs [16].

To address these issues Jamba [12] was created as a variant of both transformers and mamba SSMs that boasts a mix of efficient performance from mamba as well as enhanced context recall from transformers, by layering mamba, transformer, and mixture of experts (MOE) [15] components together at fixed ratios. The MOE component in particular allows further optimization of computational performance while also allowing for larger parameter sizes.

In this paper, I create an modified mamba architecture that aligns with the discrete spiking nature of SNNs. I then test the effectiveness of Mamba based SNNs compared to transformer based ones. I also test combined mamba-transformer models inspired by the Jamba architecture to see if I can obtain the best of both worlds.

This work is mostly focused on reducing the computational complexity of high functioning spiking neural networks while still maintaining comparable performance. The purpose of which is to create more efficient token processing framework for SNNs. Specifically, this work is designed to improve upon previous vision transformer work in the SNN space by having Mamba SSM blocks replace some of the transformer attention blocks in the network.

The primary goal of this work is to demonstrate the effectiveness and usability of selective state space models with spiking neural

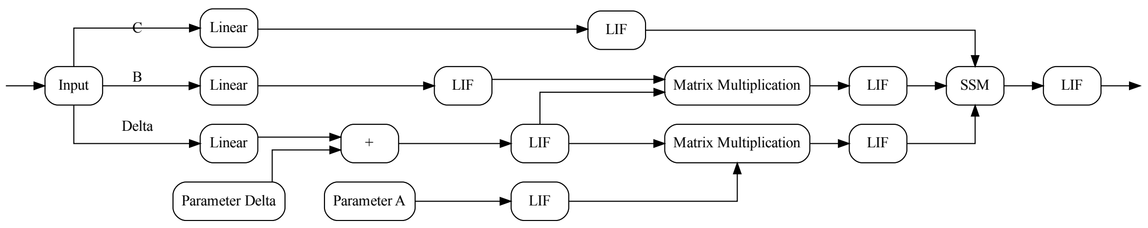

Figure 1: Spiking Mamba S6 State Selection

networks, and create a baseline for future developments in the SNN space.

3 RELATED WORK

3.1 Spikeformer [25]

The Spikeformer is one of the first attempts at making a high performance deep spiking neural network through introducing variations of attention. The primary contribution of this work is their spiking derivative of the Vanilla Self Attention, coined Spiking Self Attention (SSA). This SSA removed many of the unnecessary computation the ANN Vanilla Self Attention had, such as the softmax function and the scaling factor based on embedding length. Instead all 3 part of the query, key, and value are simply multiplied together with a single hyperparameter based scaling factor.

$$Q = \mathcal{SN}_Q(BN(XW_Q)), K = \mathcal{SN}_K(BN(XW_K)), V = \mathcal{SN}_V(BN(XW_V))$$

$$SSA = \mathcal{SN}(BN(Linear(\mathcal{SN}(QK^T V * s))))$$

In this formulation of spiking self attention, the input is converted into query, key, and value vectors which are then turned into spikes, multiplied together, turned into spikes again, then fed through a single layer of neurons to transform the values before converted back into spikes again. This formulation of Spiking Self Attention is almost the same as the one used in my work, as it is the easiest to implement and performs well enough to compare against.

3.2 Spike-driven Transformer V1 and V2 [22, 23]

The Spike-Driven Transformers both V1 and V2 are designed for vision tasks specifically. Throughout both versions, six total formulations of what they called spike driven self attention (SDSA) was created to adapt multi head self attention to a spiking compatible architecture. At first, they attempted to mirror the original vanilla transformer design, and created justifications comparing their SDSA to vanilla self attention. However, in v2 of the paper they concluded that a attention mechanism faithful to the original vanilla architecture did not lend itself to do well in the spiking environment. Therefore the final version of SDSA is formulated to be almost the exact same as the one from Spikeformer. This is another reason the spikeformer SDA design is used in this work, as all leading research in the area seem to have converged on this similar design.

3.3 Vision Transformers (ViT) [3]

Both the Spikeformer and the Spike-driven Transformer have mainly been focused on vision tasks. Vision Transformers have been popular as the latest state of the art visual content processing system, for a variety of tasks including object classification, object detection, and object segmentation. The primary structure of an VIT is 3 parts. The first part being a way to separate an image into discrete tokens rather than one entire image. This is done by generating patches of the image through a convolution [20] neural network, where the resulting channels are the tokens feed into the ViT. The second part is the attention mechanism, and the third part is the output, which is specific to each task type.

3.4 Mamba [5]

Mamba is a relatively recent development that aims to be a way around the high computational costs of transformers. They built upon the non selective structured state space model (S4) [7], and created a selective structured state space model (S6). The main idea behind structured state space models is to model long term dependencies using recurrent elements. The RNN is a type of structured state space model. Mamba's main improvement is a way to selectively focus on input tokens to pick out information compared to traditional state space models by maintaining a time dependent state. One of the main focuses of mamba implementation was a hardware aware algorithm for state selection. However, for this work, a simpler state selection mechanism is used to integrate with spiking.

3.5 Jamba [12]

With the development of Mamba, while the computational complexity issue was addressed, new issues were introduced compared to the attention mechanism in Mamba's weaker ability to recall information, especially when the amount of information needed to be copied increases. Jamba takes a best of two world's approach by introducing a hybrid architecture of transformer and mamba blocks interviewed with each other. This allows for Jamba to both have to information copying and recall of transformers as well as the computational efficiency of mamba. Specifically, Jamba used a

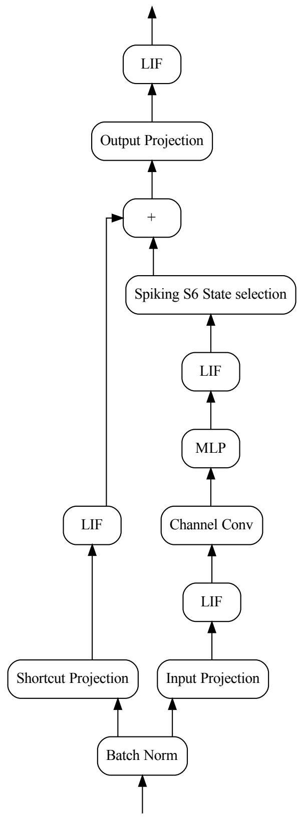

Figure 2: Spiking Mamba Block Architecture

```
Input: x : (B, L, D)
Output: y : (B, L, D)
A : (D, N) ← Parameter
B : (B, L, N)  ← Linear(x)
C : (B, L, N)  ← Linear(x)
Δ : (B, L, N)  ← Softmax(Parameter + Linear(x))
$\overline{A}$ : (B, L, D, N)  ← Softmax($\Delta A$)
$\overline{B}$ : (B, L, D, N)  ← $\Delta B$
y :  ← SSM($\overline{A}$, $\overline{B}$, C)(x)
return y
```

Figure 3: Vanilla Mamba S6 Selection

```
Input: x : (B, L, D)
Output: y : (B, L, D)
A : (D, N) ← $\mathcal{SN}$(Parameter)
B : (B, L, N)  ← $\mathcal{SN}$(Linear(x))
C : (B, L, N)  ← $\mathcal{SN}$(Linear(x))
K : (D)  ← $\mathcal{SN}$(Parameter)
Δ : (B, L, N)  ← $\mathcal{SN}$(Parameter + Linear(x))
$\overline{A}$ : (B, L, D, N)  ← $\mathcal{SN}(\Delta A)$
$\overline{B}$ : (B, L, D, N)  ← $\mathcal{SN}(\Delta B)$
y : ←  $\mathcal{SN}$(SSM($\overline{A}$, $\overline{B}$, C)(x) + $x \odot K$)
return y
```

Figure 4: Spiking Mamba S6 Selection

structure of 1 transformer block with 7 mamba blocks to introduce a 8 block architecture. Jamba also implemented a mixture of experts (MOE) system that replaced the multi layer perception parts for some of the mamba blocks to further increase efficiency. In this work, I also test interweaving spiking self attention transformer encoder blocks with spiking mamba blocks as inspired by Jamba, albeit without the MOE portion due to time constraints.

4 METHODOLOGY

4.1 Core Spiking Mamba Block structure

As shown in Figure 2, the spiking mamba block is a modified version of the original mamba block for ANNs. Notably, all activations are replaced with Leaky-Integrate and Fire activation. One other thing to note is contrary to the RMSNorm used for the original mamba architecture, which is a more optimized version of layer norm for designed to improve performance, the spiking mamba block uses the standard BatchNorm. This is due to batch norms being the type of normalization used for the spiking transformer architectures as well, and to compare and contrast the effectiveness of the two methods, I aim to minimize the amount of external difference beside the architecture itself.

The input projection used in the block is a forward projection that maps the batch normed input tokens into a higher dimensional vector space. While this does not increase the amount of tokens, it creates a bigger embedding for each token that helps with the performance of the model in the token mixing process. Here a separate projection matrix is used for the input projection and the shortcut projection. This is the same way the original mamba block for ANNs are implemented and this separation helps the model converge faster.

By default, the spiking mamba block uses an input projection size of 2 as it offered good results, with increasing the projection size not drastically changing how well the model performs. That is both the input projection layer and the shortcut projection layer maps each token from embedding size to 2 embedding size, and the output projection restores each token from 2 embedding size back to embedding size.

The Channel Conv is by default 1 layer of 1D convolution with kernel size 3 and padding 1. Similarly, the MLP is by default 1 single Linear layer. While it is possible for each of these two components to have more convolution layers and more linear layers respectively, my testing show that these factors do not effect the model performance much, and only increases the computational cost per block.

The shortcut method here is the Spike Element Wise (SEW) shortcut, similar as used in the Spikeformer. That is the shortcut being the element wise addition is done after the Leaky-Integrate and Fire layers of both the Shortcut Projection as well as the Spiking S6 State Selection layer, then the result is converted into spikes again through an LIF layer after the element wise add. While some research has been done on the difference between various shortcut types, including the Membrane Shortcut used in the Spike Driven Meta-Transformer V2, the results show only a minor difference between each shortcut type. Again for the sake of consistency in comparing against the Spikeformer's architecture on Spiking Self Attention, this work by default uses the SEW shortcut method.

The Spiking Mamba block can itself be thought of as a type of meta-transformer block, with it's token mixer being the S6 selection mechanism and the Channel Conv, MLP, and Input, Output projections thought of as the MLP section. Even though the order the layers are implemented in the Spiking Mamba block do not exactly match the structure of a meta-transformer, that is having the token mixer first then the MLP layers after, the block itself can be swappable with any other spiking meta-transformer block, including the SDA block in the Spikeformer as well as the SDSA block in the Spike-Driven Meta-Transformer. Thus further experimentation can also compare slotting the Spiking Mamba Block directly into the architectures of future works.

4.2 Spiking Selection S6 Architecture

The Selection S6 Architecture, show in Figure 1 is the heart of Mamba, similar to Multi head Self Attention in Transformers. Similar to how previous works converted vanilla self attention to spiking self attention by slightly changing the operations involved and including an LIF layer at each output, the Spiking Selection S6 is much of the same.

Primarily, every unique operation listed in Figure 4 is now given an LIF activation after performing the operation, as shown in Figure 4. Additionally, the Softmax operations on the Δ and $\overline{A}$ variables were removed for the same reason they were removed in the various spiking attention operations. That is the softmax induces additional computational cost that are unnecessary for spiking neural networks, as spikes are already non negative values at exactly 0 and 1, leading to the soft max operation being both unnecessary and ineffective.

One of the advancements for mamba is the hardware aware efficient implementation for the state space search of the SSM. For my implementation of the Spiking Mamba S6 Selection, unfortunately I did not have time to research and implement a hardware aware algorithm, instead my implementation uses a less efficient non hardware aware version that performs 2 matrix multiplications and 1 add for every token in the input.

Most of the parameters for the S6 selections are determined by the overall model, such as the batch size, number of tokens, and the embedding size, denoted as B, L, and D respectively in Figure 3 and Figure 4. The one parameter unique to the S6 selection process is an state size, denoted N, to represent the time varying internal state of the SSM mechanism. By default, this work uses an internal state of 64.

```
**Output**: y : (B, L, D)
**Input** $\overline{\mathbf{A}}$ : (B, L, D, N)
**Input** $\overline{\mathbf{B}}$ : (B, L, D, N)
**Input C** : (B, L, N)

**G** : (B, D, N) $\leftarrow$ Initalize as Zeros
$\overline{\mathbf{Y}}$: (D, L, B) $\leftarrow$ Initalize as Zeros

for i in 1...L
    **G** $\leftarrow$ $\overline{\mathrm{A}}$[:, i, :, :] $\odot G$ + $\overline{\mathrm{B}}$[:, i, :, :]
    $\overline{\mathbf{Y}}$[i, :, :]  $\leftarrow$ $(GC$[:, i, :]$^T)$[:, :, 1]
endfor

**y** $\leftarrow$ $\overline{\mathrm{Y}}^T$
**y** : $\leftarrow$  $\mathcal{SN}$(SSM($\overline{\mathrm{A}}$, $\overline{\mathrm{B}}$, C)(x))
return y
```

Figure 5: SSM

4.3 Transformer Blocks

Inspired by Jamba, this work also combines the spiking self attention with the mamba state selection mechanism by interweaving mamba blocks and transformer blocks for a balance of computational efficiency and effectiveness.

$$Q = \mathcal{SN}(BN(Conv(X)))$$

$$K = \mathcal{SN}(BN(Conv(X))$$

$$V = \mathcal{SN}(BN(Conv(X))$$

$$SSA = BN(Conv((\mathcal{SN}(QK^TV))))$$

Specifically I use a slightly modified version of the spiking self attention from Spikeformer [25], with the difference being the scaling factor s is removed, and instead the final LIF layer that process the output of the attention has a learnable threshold. This is similar to the SDSA 4 from the Spike Driven Meta-Transformer v2 [22], however without the more efficient parameterized convolution as that would take more implementation and also be more costly to train.

For the most part the transformer architecture is about the same as previous papers on spiking transformers. The main purpose of this work was not to significantly change spiking transformers rather use it in coordination with the spiking mamba block.

4.4 Spiking Patch Generator

For the ViT Portion of this work, I use a spiking patch generator that uses L Convolution blocks. Each Convolution block is shown in figure 5 as a 2D convolution of kernel size 3, followed by a 2D batch norm, then a 2D Max Pooling with kernel size 2, stride 2, and padding 1. Finally each block ends wiht an Leaky-Integrate and Fire layer to make it a part of the spiking neural network.

For each of these blocks, the input channels start at the given source input, then outputs twice as many channels as it inputs. In

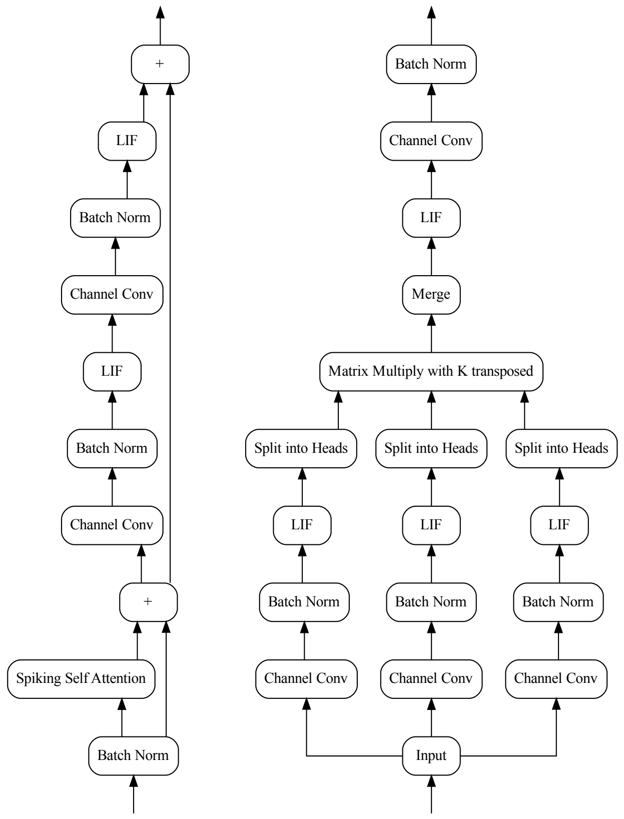

Figure 6: Spiking Transformer Encoder Block Architecture

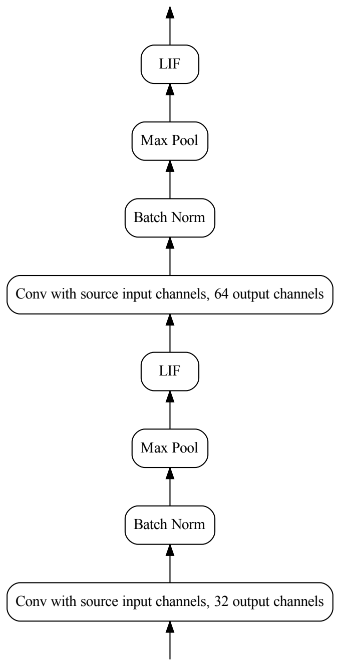
LIF
Max Pool
Batch Norm
Conv with source input channels, 64 output channels
LIF
Max Pool
Batch Norm
Conv with source input channels, 32 output channels

Figure 7: Spiking Transformer Encoder Block Architecture

otherwise, given L blocks from block number 0 to block number L-1, the output channel of each block is equal to the final output channel divided by $2^{\text{Block Number}}$. This works well and is quite standard in preexisting ViT works.

4.5 Encoding

For a spiking neural network, the encoding and decoding of input data into spikes and output data from spikes is fundamental to the functionality of the model. There are many ways to perform encoding, from the relatively simple and computationally easy methods such as latency encoding to much harder methods such as delta modulation.

4.5.1 Rate Encoding.

One of the methods used for my testing is Rate Encoding [11], which roughly take the input value, and interpret it as a spike rate. This spike rate then corresponds to the number of spikes happening within a time step.

More formally, define spikes for each time step as a random variable with probably p of occurring where p is equal to the normalized value between 0 and 1 for the real number input. For an image, this would be the normalized RGB pixel values between 0 and 1. Subsequently, each time step is defined as an independent Bernoulli trial with probability p.

The total amount of spikes within all k time steps is then defined as being sampled from a Binomial distribution with parameters k and p. This is the primary method used for the tests outside of ablation as it maintains a balance between computational efficiency and robustness.

4.5.2 Latency Encoding.

This method of latency encoding does not require the randomness of rate encoding, and thus also is computationally less expansive. Latency encoding converts each input value into a discrete integer between 1 and k, with k being the number of time steps. For each scaled input value i, the ith time step would have a spike of 1 with all other time steps containing the no spike value of 0.

This method while less computationally expansive, does lead to a very sparse amount of spikes, carrying less information at each time step. Thus it is not used as the default encoding method for testing.

4.6 Decoding

As the primary testing done in this work is categorization. Both for images in the cifar-10 and cifar-100 tests, as well as for the basic mnist hyperparameter abilation test. The model outputs a vector of length n for the n categories of the dataset for each k of the k time steps. Each output vector is a spiking vector of either 0 or 1 for off and on spikes. The selected category is determined by the maximally occurring spiking index.

$$\text{Let } k \text{ be number of time steps }, n \text{ be number of categories}$$

$$\text{Output}[K, N] \leftarrow \text{Model}(Input[K, N])$$

$$\text{Histogram} = \text{Count}(\text{Output})$$

$$\text{Result} = \text{Mode}(\text{Count})$$

4.7 Loss

Due to the decoding used here being not a traditional categorization, but rather a sort of categorization aggregating the output from k time steps, a traditional loss function as as Cross Entropy loss cannot directly be used to determine the error. Rather spiking time step aware losses have to be used.

4.7.1 *Cross Entropy Spike Count Loss.*

The Cross Entropy Spike Count [21] loss function can be thought of as a simple extension of the Cross Entropy loss function across time steps. Just as the output counts are accumulated for a histogram to determine the final categorization result, the Cross Entropy Spike Count loss sums up the counts over the time steps for each category into a histogram vector, then applies traditional Cross Entropy (Log Softmax + NLLLoss) to the resulting vector. While this method works, this is not the primary loss function used for my testing.

4.7.2 *Cross Entropy Spike Rate Loss.*

The Cross Entropy Spike Rate loss [17] function is similar to the Cross Entropy Spike Count loss function in the sense it is also a extension of the Cross Entropy loss function onto spiking neural networks with k time steps. However, the order of operations here differ from the Cross Entropy Spike Count Loss in the sense that the latter accumulate the counts for each category first before performing a Cross Entropy on the overall count vector, the Cross Entropy Spike Rate Loss instead performs Cross Entropy for each of the individual spike output vectors at each time step, then the resulting Cross Entropy loss itself is summed across all k time steps for the final loss.

This is the default loss function used for my testing as It works and is more in line with the concept of spiking neural networks. That is to handle input and outputs on a time step basis, including calculating the loss, rather than doing so on the final result.

5 RESULTS

5.1 Basic MNIST Hyperparameter Ablation

MNIST is a set of 60000, 28x28 single color images with 10 classes. This test is designed to see how well the basic mamba block performs for each parameters, in order to determine the more optimal hyperparameters to be used for the more costly and time consuming CIFAR-10 and CIFAR-100 tasks.

By nature, the parameter sizes for these model is designed to be very small, rather it is designed to test out general trends in the hyper parameters in relation to performance to determine the best values for the other tests. All trials here are run on 2 epochs of batch size 256 each. All trials are also ran with an embedding size of 64. Each combination is run 3 times with the mean of the results taken for training time and accuracy.

From my testing, increasing the number of channels drastically speed up the convergence of the model in terms of time steps taken, however, it drastically slows down the computation of the model in terms of physical time taken to train. From the mnist results, which mnist images are 28x28 single channel images, the 16 channel worked the best efficient in terms of speed of convergence with the 32, 64 channel gaining no significant increases in convergence performance while adding more computational cost. Thus I also use this result for the 3 channel datasets.

Continuing to the Time Step section, it seems that there is a big leap in accuracy between 2 and 4 time steps, however, the running time between 2 and 4 time steps didn't seem to matter much. In fact the average training time for the 2 time step version slightly exceeds the 4 time step version. I think this is mostly due to the overhead of other time consuming tasks being greater than inferencing 2 more times. From these results I decided to use 4 time steps for subsequent testing.

For the number of patchifier blocks used. The results show that 2 and 3 are the best results for mnist, both with around the same accuracy. With the 1 block version being significantly less effective, and the 4 block version dropping slightly in accuracy as well. Surprisingly, all 4 variants took around the same amount of time to train, demonstrating that the patchification process is not the most time consuming part of the model. I proceeded with 2 blocks for the rest of the ablation.

For the combination of transformer and mamba vs just 2 mamba blocks. My testing show that due to the smaller parameter size of the transformer block in this case, as the internal mamba state uses parameters of size 32. The transformer block combination was faster to train, however, also converged slower than the mamba only block.

5.2 CIFAR-10

CIFAR 10 is a set of 60000, 32x32 single color images with 10 classes. This test is designed to test the mamba block in an more challenging standard environment. To see how mamba blocks compare to past works and their resulting with spiking transformers, as well a see how mixing transformer and mamba blocks could effect the results.

While I did obtain results in this work, the results themselves are not great and do not complete with previous transformer only works. While I have tried many configurations, the training time for each configuration range from 1 to 2 days. Given the limited

Table 1: MNIST Hyper parameter Ablation

Model Size (Parameters)	Channels	Time Steps	Patchifier Block	Block Composition	State Size	Training Time (minutes)	Accuracy
150k	8	10	2	Mamba + Mamba	32	3.02	0.92
157k	16	10	2	Mamba + Mamba	32	8.24	0.99
176k	32	10	2	Mamba + Mamba	32	14.05	0.99
229k	64	10	2	Mamba + Mamba	32	36.52	0.99
157k	16	2	2	Mamba + Mamba	32	2.41	0.93
157k	16	4	2	Mamba + Mamba	32	2.32	0.98
157k	16	8	2	Mamba + Mamba	32	7.45	0.99
157k	16	10	2	Mamba + Mamba	32	8.24	0.99
165k	16	4	1	Mamba + Mamba	32	2.30	0.85
157k	16	4	2	Mamba + Mamba	32	2.32	0.98
155k	16	4	3	Mamba + Mamba	32	2.34	0.98
154k	16	4	4	Mamba + Mamba	32	2.33	0.96
157k	16	4	2	Mamba + Mamba	32	2.32	0.98
89k	16	4	2	Mamba + Transformer	32	1.81	0.85

Table 2: CIFAR-10 Results

Model Size (Parameters)	Channels	Embedding	Time Steps	Patchifier Block	Block Composition	State Size	Accuracy
17.53 M	-	-	-	-	ResNet-19 [8]	-	94.44
4.15M	-	256	4	-	Spikeformer-4-256 [25]	-	93.94
313K	128	8	4	4	4x Mamba (This)	16	63.23
3.80M	8	256	4	4	4x Mamba (This)	16	89.56
3.80M	8	256	4	4	3x Mamba + 1x Transformer (This)	16	84.60

amount of time, I was only able to achieve a maximum accuracy of 89.56 with a larger model and 63.23 for a smaller model size.

6 DISCUSSION

6.1 Training Methods

To speed up the training process, I used mixed precision training with float16 data types and "high" precision matrix multiplications that uses bloat16_3x data types. From my testing, using these 16 bit data types for training the models drastically speed up the training time by at least 1.5 to 2 times. However, the impacts whether using mix precision compromised on any effectiveness of the models.

6.2 Limitations

One major limitation I faced throughout this project is a lack of compute power, especially to make larger models or training my models faster. This compounded with the time limitations for the project and partially contributed to the lack luster results of this work.

For the architecture itself, a major limitation is the selective scan process. The selective scan process used in the implementation of this model is based on a relatively simple software algorithm that does not account for hardware efficiency. Rather the discretizations tensors of every token is looped over, and 2 matrix multiplication and 1 element wise add is used for each token. One of the main speed advantages of mamba comes from the hardware aware selective state space scanning, which is not implemented here. This could be a future extension to the project.

6.3 Future Extensions

Plenty more work can be done on this project. Both in terms of accomplishing the purpose of this paper, that is create a mamba based spiking vision network that can compete with existing spiking ResNet and spiking transformer based ones. To this end, obtaining > 90% results on CIFAR-10 for the 3 to 5 Million parameter size, and obtaining close to 100% results on greater parameter sizes is a must.

CIFAR-100 and ImageNet-1K were originally planned for this work. However due to time limitations and my inability to obtain state of the art results on CIFAR-10 stumped the progress of the CIFAR-100 and ImageNet-1K. The furthest I've gotten on CIFAR-100 is with a similar 3.8M parameter mamba model, at a 20% accuracy after 3 days of training. I believe it is possible to achieve better results using the same or some modifications to the Spiking Mamba blocks in this paper with more time and resources.

In the vision domain, tasks such as Image Segmentation, Object detection, and Neuromorphic datasets are all tested by other SNN focused papers that spiking mamba could be applied to. Outside of the vision domain, an auto regressive version of the spiking mamba model could potentially be used for other fields such as text, image, audio generations.

7 CONCLUSION

In this work, I adapted the Mamba block into a Spiking Mamba block, with the heart of mamba, the S6 Selection mechanism also implemented as an Spiking S6 Selection module. In addition, I adapted a variation of a spiking patch generator, as well as attempted to mix Spiking Mamba blocks with Spiking Transformer blocks containing a slightly modified version of the Spiking Self Attention as inspired by Jamba.

While the results obtained in this work are not quite state of the art. I have gotten close with a close to 90% accuracy on the CIFAR-10 dataset. In addition, I have also determined some patterns in the hyper parameters from the Ablation that demonstrated the effects and side effects of hyper parameters that are not easily intuitive without testing.

I believe combining mamba with SNNs is one of the better ways to move spiking neural networks forward rather than focusing on more elaborate convolution and creating more types of spiking self attention.

REFERENCES

[1] Sayeed Shafayet Chowdhury, Nitin Rathi, and Kaushik Roy. 2021. One Timestep is All You Need: Training Spiking Neural Networks with Ultra Low Latency. arXiv:2110.05929 [cs.NE]

[2] Wojciech Marian Czarnecki, Grzegorz Świrszcz, Max Jaderberg, Simon Osindero, Oriol Vinyals, and Koray Kavukcuoglu. 2017. Understanding Synthetic Gradients and Decoupled Neural Interfaces. arXiv:1703.00522 [cs.LG]

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929 [cs.CV]

[4] R Farsani, Ehsan Pazouki, and Jecei Jecei. 2021. A Transformer Self-Attention Model for Time Series Forecasting. *Journal of Electrical and Computer Engineering Innovations* 9 (01 2021), 1–10. https://doi.org/10.22061/JECEI.2020.7426.391

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

[6] Albert Gu and Tri Dao. 2024. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. arXiv:2312.00752 [cs.LG]

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

[8] Yifan Hu, Lei Deng, Yujie Wu, Man Yao, and Guoqi Li. 2021. Advancing Spiking Neural Networks towards Deep Residual Learning. https://doi.org/10.48550/ARXIV.2112.08954

[9] Samy Jelassi, David Brandfonbrener, Sham M. Kakade, and Eran Malach. 2024. Repeat After Me: Transformers are Better than State Space Models at Copying. arXiv:2402.01032 [cs.LG]

[10] Feyza Duman Keles, Pruthuvi Mahesakya Wijewardena, and Chinmay Hegde. 2022. On The Computational Complexity of Self-Attention. arXiv:2209.04881 [cs.LG]

[11] Youngeun Kim, Hyoungseob Park, Abhishek Moitra, Abhiroop Bhattacharjee, Yeshwanth Venkatesha, and Priyadarshini Panda. 2022. Rate Coding or Direct Coding: Which One is Better for Accurate, Robust, and Energy-efficient Spiking Neural Networks? arXiv:2202.03133 [cs.NE]

[12] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida, Amir Bergman, Roman Glozman, Michael Gokhman, Avashalom Manevich, Nir Ratner, Noam Rozen, Erez Shwartz, Mor Zusman, and Yoav Shoham. 2024. Jamba: A Hybrid Transformer-Mamba Language Model. arXiv:2403.19887 [cs.CL]

[13] Wolfgang Maass. 1997. Networks of spiking neurons: The third generation of neural network models. *Neural Networks* 10, 9 (Dec. 1997), 1659–1671. https://doi.org/10.1016/s0893-6080(97)00011-7

[14] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. 2019. Towards spike-based machine intelligence with neuromorphic computing. *Nature* 575, 7784 (Nov. 2019), 607–617. https://doi.org/10.1038/s41586-019-1677-2

[15] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. arXiv:1701.06538 [cs.LG]

[16] Guobin Shen, Dongcheng Zhao, Tenglong Li, Jindong Li, and Yi Zeng. 2023. Is Conventional SNN Really Efficient? A Perspective from Network Quantization. arXiv:2311.10802 [cs.NE]

[17] Sumit Bam Shrestha, Longwei Zhu, and Pengfei Sun. 2022. Spikemax: Spike-based Loss Methods for Classification. arXiv:2205.09845 [cs.NE]

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL]

[19] Zihan Wang, Fanheng Kong, Shi Feng, Ming Wang, Xiaocui Yang, Han Zhao, Daling Wang, and Yifei Zhang. 2024. Is Mamba Effective for Time Series Forecasting? arXiv:2403.11144 [cs.LG]

[20] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. 2021. CvT: Introducing Convolutions to Vision Transformers. arXiv:2103.15808 [cs.CV]

[21] Jibin Wu, Yansong Chua, Malu Zhang, Qu Yang, Guoqi Li, and Haizhou Li. 2019. Deep Spiking Neural Network with Spike Count based Learning Rule. arXiv:1902.05705 [cs.NE]

[22] Man Yao, JiaKui Hu, Tianxiang Hu, Yifan Xu, Zhaokun Zhou, Yonghong Tian, Bo XU, and Guoqi Li. 2024. Spike-driven Transformer V2: Meta Spiking Neural Network Architecture Inspiring the Design of Next-generation Neuromorphic Chips. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=1SIBN5Xyw7

[23] Man Yao, Jiakui Hu, Zhaokun Zhou, Li Yuan, Yonghong Tian, Bo Xu, and Guoqi Li. 2023. Spike-driven Transformer. arXiv:2307.01694 [cs.NE]

[24] Chenlin Zhou, Han Zhang, Liutao Yu, Yumin Ye, Zhaokun Zhou, Liwei Huang, Zhengyu Ma, Xiaopeng Fan, Huihui Zhou, and Yonghong Tian. 2024. Direct Training High-Performance Deep Spiking Neural Networks: A Review of Theories and Methods. arXiv:2405.04289 [cs.NE]

[25] Zhaokun Zhou, Yuesheng Zhu, Chao He, Yaowei Wang, Shuicheng Yan, Yonghong Tian, and Li Yuan. 2022. Spikformer: When Spiking Neural Network Meets Transformer. arXiv:2209.15425 [cs.NE]

[26] Itamar Zimerman and Lior Wolf. 2023. On the Long Range Abilities of Transformers. arXiv:2311.16620 [cs.LG]