Capabilities and Applications of Deep Learning Recurrent Models

Data Science and Engineering

Destiny Jarymaya Okpekpe

A.a. 2023/2024

Advisors:
Prof. Dr. Lia Morra, Politecnico di Torino
Prof. Dr. Thomas Hofmann, ETH Zürich
Dr. Antonio Orvieto, Max Planck Institute for Intelligent Systems & ELLIS Institute

Abstract

Even with the major advances in Language Modeling (LM) in recent years after the introduction of transformer architecture, reasoning is still one of the unique skills of the human brain that Deep Learning models struggle to replicate the most.

Since one of the main challenges is to efficiently recall information seen in the past, the Associative Recall (AR) synthetic task has gained importance for being a good proxy for language modeling and a suitable benchmark to select promising Large Language Models (LLM). A series of recurrent-gated models (such as H3, Mamba and Hyena), built to overcome the drawbacks of the $O(L^2)$ computational complexity of the attention module, recently gained popularity for solving AR even with long sequences ($\geq 10,000$ tokens). However, when scaled and trained on real language tasks, these models still cannot achieve the performance of transformers. In the first part of this work, I investigate the reason for this gap and find three main components responsible for it: (1) the fact that AR is not challenging enough to be a proxy for language, (2) the fact that recurrent models deeply relies on proper optimization to efficiently update their hidden state and (3) the fact that while transformers benefit the most from scaling in depth, recurrent models benefit the most from scaling in width.

When reasoning with sequences, another difference between transformers and recurrent models is the role of Positional Embedding (PE), since in the latter models the relative position of tokens is given by their order in the sequence (implicit causality). The question is how to reconcile data modalities that are not sequential, such as in the 3D Vision domain, with the inherently directional (or bi-directional) order-dependent processing of recurrent models like Mamba. In the second part of this work, I introduce a method to convert point clouds into 1D sequences that maintain 3D spatial structure with no need for data replication, allowing Mamba's sequential processing to be applied effectively in an almost permutation-invariant manner. In contrast to other works, this method does not rely on positional embeddings and does not replicate the input sequence length while still surpassing Transformer-based models in both accuracy and efficiency.

Contents

Chapter 1

Introduction

When thinking about intelligence and what differentiates us from animals, one of the main features of the human brain that comes to mind is the ability to reason and to produce language.

In recent years, major improvements in the field of Natural Language Processing (NLP) were led by the introduction of the transformer architecture ([1]), making it the de facto standard for language processing. However, such models come with an $O(L^2)$ complexity bottleneck, where L is the length of the sequence feed to the model, making it impossible to apply the transformer architecture with data modalities that have extremely long sequences inherently, such as in 3D vision and DNA sequencing. To overcome this drawback, a new category of models where introduced, such as State-Space Models ([2], [3], [4]), RWKV ([5]) and Hyena ([6]), that use recurrence, gating and convolution to reach the performances of transformers while still maintaining linear complexity.

However, since training in language is an expensive task, it is becoming popular to preliminary test new models' capabilities on simpler synthetic tasks that try to approximate language modeling (proxy). This is the case of the Associative Recall (AR) task, which tries to mimic the ability to recall tokens already seen in the past. As shown in previous works such as [2], [3], [4], these recurrent models are extremely proficient in solving AR, especially when the sequence length is in the order of tens of thousands of tokens. However, these models still are not able to reach the performances (in terms of perplexity) of transformers when trained in proper language (i.e. [7]).

In order to investigate better such behavior, my thesis is structured in the following way:

In Chapter 2, I give some background on what is reasoning, how the Associative Recall (AR) task is used to test capabilities of Deep Learning Models, and finally I show some of the most common and promising architectures used in this field.

In Chapter 3, I present the work that inspired my thesis, Zoology ([7]), and highlight its main drawbacks.

In Chapter 4, following the work of Zoology, I investigate the differences in the reasoning capabilities of attention and recurrent models, leading to the following contributions: (1) Finding a more challenging version of the AR task that betters approximate language modeling, (2) understand the role of optimization in recurrent models and how crucial a proper learning rate is for these models to efficiently compress information in their hidden state, and (3) how attention and recurrent models should be scaled differently, since the first benefits the most when scaled in depth, whereas the others benefit the most when scaled in width.

In Chapter 5, after understanding how the different nature of attention and recurrent model impact their capabilities, I test these differences in a field different from language. Specifically, I use a recurrent model in a 3D computer vision application, a domain that usually prefers set operations mechanism like attention. This work led to the following contributions: (1) Reconcile the recurrent nature of the selective State-Space Model (SSM) Mamba with point clouds, a data modality that is inherently a set, and (2) understand the role of positional embeddings with causal models such as SSMs.

Finally, in Chapter 6, I discuss my conclusion and future works.

Chapter 2

Background

2.1 Reasoning and Associative Recall

Natural Language Processing (NLP) is becoming an increasing central fields in Artificial Intelligence (AI), especially after the introduction of the transformer architecture [1]. The standard procedure is to build a Large Language Model (LLM) which is trained on producing text autoregressively: the goal is to build a model that, given a context, outputs a distribution over the words in the vocabulary. More formally, given a vocabulary of natural language tokens, Language Modeling (LM) is the task of estimating a probability distribution over all possible sequences of tokens from that vocabulary. Let V be a vocabulary of natural language tokens and let $y = (y_1, y_2, ..., y_T), y_i \in V$ be a sequence of natural language tokens, a language model estimates the probability $p(y)$ over all possible sequences of token with the following:

$$p(y) = p(EOS|y) \prod_{t=1}^{T} p(y_t|y_{<t})$$

where EOS is a special end-of-sequence token.

To estimate the quality of a language model, which is determined by how well it approximates the probability of token given text sequence (context), it is not recommended to use a binary metric such as accuracy. Instead, I present here the definition of *perplexity*, a preferred metric to evaluate LLM performance. One can think of this as minimizing the cross-entropy H of training samples that were drawn from the empirical distribution p. The entropy of the language model p on a given sequence of tokens $y = (y_1, y_2, ..., y_T) \in V$ reads:

$$H(p) = -\frac{1}{T} \sum_{t=1}^{T} \log(p(y_t|y_{<t}).$$

Here, the quantity $-\log(p(y_t|y_{<t})$ can be interpreted as a measure of how the model is "surprised" to see a specific token. If the model assigns a very low probability to the sample, the measure of surprise will be large. This indicates that the model could not estimate adequately the words distribution. If instead $p(y_t|y_{<t})$ is close to one, the logarithm will be almost zero and this indicates that the model was not surprised by the sample. The entropy defines the average number of bits per token needed to encode it. Therefore, the exponential of the entropy represents the average number of choices the random variable has. For example, for a synthetic language with V words, if all words can occur in each processing step with uniform probability, then the entropy is $\log(V)$ and the exponential of the entropy is exactly the overall number of tokens. If however, only a subset of size k of the tokens can occur in each step, then the entropy will be $\log(k)$ and the number of choices is exactly k. The exponential of the entropy is called *perplexity*.

However, even after all the advances in language modeling, there is still a lack of actual reasoning capabilities in Deep Learning models, resulting in a still noticeable difference in performances between humans and these models. This gap becomes even more noticeable, as an example, when we ask models to solve mathematical problems in which given an input, there is a need to produce multiple intermediate reasoning steps before being able to properly give a correct answer.

One of the main language skills that humans have is the ability to efficiently recall information already seen in the past. Specifically, given the sentence:

*"**Hakuna Matata**"! It means **no worries** for the rest of your days"*

when facing again the words *"**Hakuna Matata**"*, we expect a reasonable and intelligent agent to easily (with higher probability) recall the words *"**no worries**"*.

Based on this idea, the Associative Recall (AR) synthetic task ([8], [9], [10]) gained more and more relevance for being a suitable proxy for language modeling in the first place, but also for being a task not so resource exhausting compared to proper language modeling. More formally, in the AR task, we give as input a sequence of key-value pairs (letters and numbers in this example) from a vocabulary

$$A\ 6\ I\ 9\ C\ 7\ P\ 1\ S\ 4\ D\ 2$$

and we ask the model to recall the value of a key already seen before in the sequence

$$C \rightarrow\ ?$$

which in this case is 7.

Since performing associative recall is akin to reasoning, we expect that models capable of solving such task are promising LLM when scaled and trained on text datasets. In the next sections, I will introduce a list of famous and promising models used to solve specifically the associative recall task and that show some degree of reasoning capabilities.

2.2 Attention Mechanism and Transformers

The Transformer model, introduced in [1], represents a groundbreaking approach to machine translation and more generally to Natural Language Processing (NLP). It departs from the recurrent and convolutional architectures that were standard in sequence transduction tasks by using a model entirely based on the attention mechanisms to draw global dependencies between input and output. By doing so, transformers achieve superior performance and parallelization during training compared to previous state-of-the-art models. Previously, most competitive neural sequence models had an encoder-decoder structure, where the encoder maps an input sequence $(x_1, ...x_n)$ of symbols to a sequence of continuous representation $\mathbf{z} = (z_1, ...z_n)$. Then, given $\mathbf{z}$, the decoder generates an output sequence of symbols one element at a time, in an autoregressive manner, using the previously generated symbols as additional input.

In the transformer architecture, both encoder and decoder can be described by 3 main components:

- **The attention module**: this is the main component of the transformer architecture that allows each token in the sequence to attend to all the others in order to predict the next token in the sequence. Firstly, the input sequence u is projected by the learnable matrices W_Q, W_K and W_V to the query, key and value vectors by the following:

$$q = uW_Q, \qquad k = uW_K, \qquad v = uW_V$$

  then the key, query and values vectors are combined and scaled to obtain the matrix output with the following:

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

  This module is also called sequence-mixer because it allows an exchange of information (mixing) between the tokens in the input sequence;

- **Position-wise Feed-Forward Networks**: a standard Multilayer Perceptron (MLP) applied to all the tokens in the sequence independently in the following way:

$$\texttt{FFN}(x) = \texttt{max}(0, xW_1 + b_1)W_2 + b_2$$

This module is also called channel-mixer because it allows an exchange of information (mixing) across the features dimension (channels) of the tokens independently of the other tokens in the sequence;

- **Positional encoding**: since the attention mechanism is a set operation, to give information about the order in the sequence, this particular positional embedding is added to the input. In the original work [1], sine and cosine functions were used to simulate such positional information:

  $$\texttt{PE}_{(pos,2i)} = \texttt{sin}(pos/10000^{2i/d_{model}})$$

  $$\texttt{PE}_{(pos,2i+1)} = \texttt{cos}(pos/10000^{2i/d_{model}})$$

  where *pos* is the position of the token in the sequence and i is the dimension, but several other proposals became popular afterwards such as Rotatory Positional Embedding (RoPE) in [11].

It is crucial to highlight that the transformer (and generally the attention mechanism) explicitly computes all the interactions between all the tokens in the sequence, resulting in a squared complexity wrt the length of the sequence processed. This will be the main drawback of the architecture that will inspire the other models described later in this work.

2.3 State Space Models

To tackle the quadratic bottleneck of the softmax function and the poor performance on long-range ($\geq 10,000$ steps) dependencies of transformers, a promising novel approach was proposed by simulating the fundamental State-Space Model (SSM):

$$x'(t) = \mathbf{A}x(t) + \mathbf{B}u(t)$$

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

where u is the input, x is the hidden state and y is the output. Such models were already used in fields such as control theory and computational neuroscience, but to use them in sequence modeling the main challenge is to properly parametrize the state matrix A that would otherwise require prohibitive computation and memory resources.

2.3.1 S4

Introduced in [3], Structured State-Space (S4) is the first efficient and performing implementation of SSM theory in sequence modeling, being the first to solve the hardest task in the Long-Range Arena (LRA) benchmark [12] where transformers always performed poorly or encountered memory error. The main contribution in S4 is the efficient parametrization of the state matrix A by decomposing it as the sum of a low-rank and normal term. Additionally, instead of expanding the standard SSM in coefficient space, its truncated generating function is computed in the frequency space, simplifying it as a multipole-like evaluation. With these two ideas, S4 results in $\widetilde{O}(N+L)$ computation and $O(N+L)$ memory usage, where N is the state dimension and L is the sequence length, which is essentially tight for sequence models.

Formally, S4 is capable of efficiently compute and parallelize the four properties of SSMs:

- **Continuous-time latent representation of SSMs**: The 1-D input signal $u(t)$ is mapped to an N-D latent state $x(t)$ before projecting it to a 1-D output signal $y(t)$ with the following system of equations:

  $$x'(t) = \mathbf{A}x(t) + \mathbf{B}u(t)$$

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

  The matrices A, B, C, D are parameters learned by gradient descent as usual. In particular, we can assume $\mathbf{D} = 0$ and consider it as an easy to compute skip connection;

- **HiPPO matrix to address Long-range dependencies**: Do to its sequential nature, SSMs also suffer from vanishing and exploding gradient problem [13]. In [14], a specific initialization matrix, called the HiPPO matrix and described as:

  $$\textbf{HiPPO Matrix} \qquad \mathbf{A}_{nk} = -\begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n \geq k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n \leq k \end{cases}$$

  has demonstrated to increase the performance of SSMs compared to a random initialized matrix A;

- **Recurrent representation with discrete-time SSM**: To apply SSM theory, we need to convert the continuous-time $u(t)$ representation to a decrete-time one $(u_0, u_1, ...)$. To do so, S4 introduces a step size Δ to approximate the state matrix A into:

  $$x_k = \overline{\mathbf{A}}x_{k-1} + \overline{\mathbf{B}}u_k$$

$$y_k = \overline{\mathbf{C}} x_k$$

where:

$$\overline{\mathbf{A}} = (\mathbf{I} - \Delta/2 \cdot \mathbf{A})^{-1}(\mathbf{I} + \Delta/2 \cdot \mathbf{A})$$
$$\overline{\mathbf{B}} = (\mathbf{I} - \Delta/2 \cdot \mathbf{A})^{-1}\Delta\mathbf{B}$$
$$\overline{\mathbf{C}} = \mathbf{C}$$

The discrete-time SSM is now just a function of the matrix A and the step size Δ;

- **Convolutional representation to train SSM**: The previous representation is not practical for training on modern hardware such as GPUs that leverage parallel computing. To solve this issue, S4 uses the well-known connection between Linear Time-Invariant (LTI) SSMs and continuous convolutions. In particular, by unrolling the sequence and considering $x_{-1} = 0$ we obtain:

$$x_0 = \overline{\mathbf{B}} u_0 \qquad x_1 = \overline{\mathbf{AB}} u_0 + \overline{\mathbf{B}} u_1 \qquad x_2 = \overline{\mathbf{A}^2\mathbf{B}} u_0 + \overline{\mathbf{AB}} u_1 + \overline{\mathbf{B}} u_2 \qquad ...$$

$$y_0 = \overline{\mathbf{CB}} u_0 \qquad y_1 = \overline{\mathbf{CAB}} u_0 + \overline{\mathbf{CB}} u_1 \quad y_2 = \overline{\mathbf{CA}^2\mathbf{B}} u_0 + \overline{\mathbf{CAB}} u_1 + \overline{\mathbf{CB}} u_2 \ ...$$

This can be vectorized as:

$$y_k = \overline{\mathbf{CA}^k\mathbf{B}} u_0 + \overline{\mathbf{CA}^{k-1}\mathbf{B}} u_1 + ... + \overline{\mathbf{CAB}} u_{k-1} + \overline{\mathbf{CB}} u_k$$

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

where:

$$\overline{\mathbf{K}} = (\ \overline{\mathbf{CB}}, \overline{\mathbf{CAB}}, ..., \overline{\mathbf{CA}^{L-1}\mathbf{B}})$$

where $\overline{\mathbf{K}}$ can be very efficiently computed with Fast Fourier Transform algorithm (FFTA).

2.3.2 H3

Introduced in [4], Hungry Hungry Hippo (H3) is a new SSM-based layer designed to solve language modeling tasks to reduce the performance gap between SSMs and transformers. To do so, H3 stacks two SSMs, with multiplicative interactions between their input and output projections. The SSMs allow H3 to easily recall tokens already seen in the sequence, while the multiplicative interactions allow comparisons across tokens in the sequence.

Next, to reduce the gap between attention and SSMs, H3 improves its efficiency on modern hardware by introducing FlashConv, a hierarchical algorithm for computing SSMs. The technical challenge is that SSMs require an FFT-based convolution over the input sequence, which requires an FFT, pointwise multiply, and inverse FFT. This operation incurs in expensive GPU memory reads/writes, and cannot utilize the specialized matrix multiply units available on modern hardware. To use specialized matrix multiply units, the authors appeal to classical techniques that split the FFT into blocks and compute it using a series of matrix multiplications. Combined with kernel fusion, this "block" FFT solution increases hardware efficiency, but only up to sequence length of 8K on modern A100. To scale to sequences longer than 8K, H3 uses a state-passing algorithm specialized to SSMs where the key insight is to leverage the recurrent properties of SSMs to process the input in chunks. The state-passing algorithm splits the input into the largest chunks that can fit into GPU SRAM, efficiently computes the FFT-based convolution using block FFT and updates an intermediate state to start the next chunk. In this way, FlashConv can scale SSMs to any sequence length while still maintaining a near linear compute complexity.

More formally, H3 can be described by the following four main components:

- **Multiplicative Interaction**: To remember tokens from the past, we want the state x_i to copy from the input u_i, and then pass that information to the next state x_{i+1}. To compare tokens across the sequence, the output of an SSM, containing information from previous time steps, is multiplied by the input at the current time step, thus measuring the similarity between tokens;
- **Recurrence**: H3 is loosely inspired by linear attention [15], where the input u is projected to get three signals $\mathbf{Q}, \mathbf{K}, \mathbf{V}$. Then, the non-linearity $\varphi(\mathbf{K})$ is replaced with an SSM where $\mathbf{A}$ is a shift matrix (SSM_{shift}) and the summation S_i is replaced with an SSM with diagonal $\mathbf{A}$ (SSM_{diag}). For the case of head dimension $d_h = 1$, the output is:

$$\mathbf{Q} \odot \mathtt{SSM}_{diag}(\mathtt{SSM}_{shift}(\mathbf{K}) \odot \mathbf{V})$$

  where $\odot$ denotes pointwise multiplication. This form can be viewed as stacking two SSMs with multiplicative interaction (each is a "hungry hippo", hence the name of the layer);
- **Remembering Key Tokens**: The shift and diagonal SSMs are designed to address the capability to log tokens after particular events. In the shift SSM, $\mathbf{A}$ is constrained to:

$$\mathbf{A}_{i,j} = \begin{cases} 1 & \text{for } i-1=j \\ 0 & \text{otherwise} \end{cases}$$

  The action of this matrix on the hidden state x_i is to shift each coordinate down by one, thereby creating a "memory" of the previous states. The diagonal SSM constrains $\mathbf{A}$ to be diagonal and initializes it from the diagonal version of HiPPO ([16]). This parameterization allows the model to remember the state over the entire sequence. The shift SSM can detect when a particular event occurs, and the diagonal SSM can remember a token afterwards for the rest of the sequence;

- **Multiplicative Interaction for Comparison**: H3 takes multiplicative interactions from linear attention [15], but provides another missing capability when combined with a shift matrix: comparing tokens across the sequence. The multiplicative interactions between the output of the shift SSM and the **V** projection mimic local multiplicative interactions in linear attention (depending on the size of the hidden state). Similarly, multiplicative interactions with the **Q** projection and the output of the diagonal SSM allow comparisons between tokens over the entire sequence.

The H3 layer can be used to construct a model in the same style as Transformers by interleaving it with MLPs, connected by residual connection and layer normalization.

2.3.3 Mamba S6

Introduced in [2], Mamba (S6) is a State-Space Model (SSM) developed to address the gap in performance between transformers and SSMs on important modalities such as language, where the key weakness of the latest is their inability to perform content-based reasoning. Mamba proposes several improvements: firstly by simply letting the SSM parameters be functions of the input, allowing the model to selectively propagate or forget information along the sequence length dimension depending on the current token. Secondly, since this change prevents the use of efficient convolutions as in S4 [3], Mamba implements a hardware-aware parallel algorithm in recurrent mode. These selective SSMs are integrated into a simplified end-to-end neural network architecture without attention, forming the Mamba architecture.

The architecture idea comes from the fundamental problem of sequence modeling, which is compressing context into a smaller state. The tradeoffs of popular sequence models can be understood in this way: for example, attention is both effective and inefficient because it explicitly does not compress context at all. This comes from the fact that autoregressive inference requires to store explicitly the entire context (i.e. the **KV** cache), which directly causes the slow inference and training quadratic-time of Transformers. On the other hand, recurrent models are efficient because they have a finite state, implying constant-time inference and linear-time training. However, their effectiveness is limited by how well the contex is compressed in the hidden state. The efficiency vs. effectiveness tradeoff of sequence models is characterized by how well they compress information into their hidden state: efficient models must have a small state, while effective models must have a state that contains all necessary information from the context.

The authors propose that a fundamental principle for building sequence models is **selectivity**: the context-aware ability to focus on or filter out inputs into a sequential state. In particular, a selection mechanism controls how information propagates or interacts along the sequence dimension. One method of incorporating a selection mechanism into models is by letting their parameters that affect interactions along the sequence (e.g. the recurrent dynamics of an RNN or the convolution kernel of a CNN) to be input-dependent. In Mamba, the main difference is simply making the parameters $\Delta, \mathbf{B}, \mathbf{C}$ functions of the input, along with the associated changes to tensor shapes throughout. In particular, these parameters now have a length dimension L, meaning that the model has changed from Linear Time-Invariant (LTI, as all SSMs) to Linear Time-Varying. Specifically:

$$s_B(x) = \texttt{Linear}_n(x)$$

$$s_C(x) = \texttt{Linear}_n(x)$$

$$s_\Delta(x) = \texttt{Broadcast}_D(\texttt{Linear}_1(x))$$

$$\tau_\Delta = \texttt{softplus}$$

where $Linear_d$ is a parameterized projection to dimension d, where the choice of s_Δ and τ_Δ is due to the connection to RNN gating mechanisms:

$$g_t = \sigma(\texttt{Linear}(x_t))$$

$$h_t = (1 - g_t)h_{t-1} + g_t x_t$$

The selection mechanism is designed to overcome the limitations of LTI models, but this change does not allow the pre-computation trick of S4 since the matrices **B** and **C** are not constant in time. Mamba addresses this problem with three classical techniques: kernel fusion, parallel scan, and recomputation.

The main idea is to leverage the properties of modern accelerators (GPUs) to materialize the state h only in the levels of the memory hierarchy that are more efficient. In particular, most operations (except matrix multiplication) are bounded by memory bandwidth and this includes SSMs scan operation. So, instead of preparing the scan input $(\overline{\mathbf{A}}, \overline{\mathbf{B}})$ of size $\mathbf{B}, \mathbf{L}, \mathbf{D}, \mathbf{N}$ in GPU HBM (high-bandwidth memory), the SSM parameters $(\Delta, \mathbf{A}, \mathbf{B}, \mathbf{C})$ are loaded directly from slow HBM to fast SRAM, the discretization and recurrence are performed in SRAM, and then the final outputs of size $(\mathbf{B}, \mathbf{L}, \mathbf{D})$ is written back to HBM. The author noticed that, despite being linear, the sequential recurrence could still be parallelized with a work-efficient parallel scan algorithm. Finally, to avoid saving the intermediate states, which are necessary for backpropagation, Mamba carefully applies the classic technique of recomputation to reduce the memory requirements: the intermediate states are not stored but recomputed in the backward pass when the inputs are loaded from HBM to SRAM. As a result, the fused selective scan layer has the same memory requirements as an optimized transformer implementation with FlashAttention [17].

The effect of the selection mechanism can be summarized in the following three main ideas:

- **Variable Spacing**: Selectivity allows filtering out irrelevant noise tokens that may occur between inputs of interest, which occurs ubiquitously in common data modalities. This happens particularly for discrete data, like the presence of language fillers such as "um". This property arises because the model can mechanistically filter out any particular input x_t;
- **Filtering Context**: It has been empirically observed that many sequence models do not improve with longer context, despite the principle that more context should lead to strictly better performance. An explanation is that many sequence models cannot effectively ignore irrelevant context when necessary. An intuitive example is global convolutions (and general LTI models). On the other hand, selective models can simply reset their state at any time to remove extraneous history, and thus their performance in principle improves monotonically with context length;
- **Boundary Resetting**: In settings where multiple independent sequences are stitched together, Transformers can keep them separate by instantiating a particular attention mask, while LTI models will bleed information between the sequences. Selective SSMs can also reset their state at boundaries (e.g. $\Delta_t \to \infty$).

Here instead I focus on the interpretation of particular parameters in Mamba:

- **Interpretation of** Δ: In general, Δ controls the balance between how much to focus or ignore the current input x_t. Mechanically, a large Δ resets the state h and focuses on the current input x, while a small Δ persists the state and ignores the current input. SSMs can be interpreted as a continuous system discretized by a timestep Δ, and in this context, the intuition is that large $\Delta \to \infty$ represents the system focusing on the current input for longer (thus "selecting" it and forgetting its current state) while a small $\Delta \to 0$ represents a transient input that is ignored;
- **Interpretation of** A: While the $\mathbf{A}$ parameter could also be selective, it ultimately affects the model only through its interaction with Δ via

$\overline{\mathbf{A}} = exp(\Delta\mathbf{A})$ (similarly to what was said before in S4). Thus selectivity in Δ is enough to ensure selectivity in $(\overline{\mathbf{A}}, \overline{\mathbf{B}})$, and is the main source of improvement;

- **Interpretation of** B **and** C: The most important property of selectivity is filtering out irrelevant information so that a sequence model's context can be compressed into an efficient state. In a SSM, modifying **B** and **C** to be selective allows finer-grained control over whether to let an input x_t into the state h_t , or the state into the output y_t . These can be interpreted as allowing the model to modulate the recurrent dynamics based on content (input) and context (hidden states) respectively.

2.4 Reccurent Gated-Convolution Models

Inspired by the success of SSMs, other models started to leverage convolutions and gating mechanisms in their recurrence to solve the quadratic complexity of the softmax function in the attention mechanism while still performing comparably to transformers.

2.4.1 Hyena

Introduced in [6], Hyena is a model that interleaves implicitly parametrized long convolution and data-controlled gating to process long sequences of data efficiently. Taking inspiration from [10], [4], in which it was suggested that the attention mechanism only utilizes a small portion of its quadratic complexity when applied in language processing, the authors questioned if it was possible to achieve the quality of attention at scale with sub-quadratic operators. To do so, Hyena is defined as a recurrence of two efficient subquadratic primitives: a long implicit convolution and element-wise multiplicative gating.

By mapping each step in the Hyena recurrence to its corresponding matrix form, the model can be defined as a decomposition of a data-controlled matrix, i.e. a matrix whose entries are functions of the input. To make this process efficient, Hyena relies on fast convolution algorithms that do not materialize the full matrix.

The main steps of Hyena can be summarized in the following way:

- **Input Projections**: First, a set of N+1 linear projections of the input are computed $(v_t, x_t^1, ..., x_t^N)$, where one of the projections takes the role of value, such that a linear input-output function can be defined as $y = H(u)$ for some $H(u)$;

- **Long Convolutions**: Then, the matrix $H(u)$ can be described by interleaving a series of implicit long convolutions and element-wise multiplication with one projection x^i at a time. All of this is done without materializing $H(u)$, obtaining an efficient data-controlled operator as a factorization of a matrix.

Formally, for an Order-N Hyena Operator, let $(v, x^1, ..., x^N)$ be projections of the input and let $(h^1, ..., h^N)$ be a set of learnable filters. The $Hyena_N$ operator is defined by the recurrence:

$$\begin{aligned} z_t^1 &= v_t \\ z_t^{n+1} &= x_t^n (h^n * z^n)_t \qquad n = 1, ..., N \\ y_t &= z_t^{N+1} \end{aligned}$$

Noticeably, Hyena can be also seen as a generalization of H3, where the number of projections is three, for an arbitrary number of projections.

2.4.2 RWKV

Receptance Weighted Key-Value (RWKV) is a model architecture introduced in [5] that combines the efficient parallelizable training of transformers and the inference efficiency of RNN. The motivation behind RWKV is to balance computational efficiency with expressive capacity in Neural Networks by alleviating the memory bottleneck and quadratic scaling of Transformers. To do so, the model reformulates the attention mechanism with a variant of linear attention, replacing the traditional dot-product token interaction with channel-directed attention. In this way, RWKV can be formulated as a Transformer during training, allowing for parallelizable training and efficient scaling, and as an RNN during inference, achieving constant computational and memory complexity.

The RWKV model architecture is defined by four fundamental elements that are responsible for the time-mixing and channel-mixing blocks:

- **R**: The Receptance vector that acts as the receiver of past information;
- **W**: The Weight acts as the positional weight decay vector, a trainable parameter. Intuitively, we want past tokens to be (generally) less relevant as time goes on;
- **K and V**: The Key and the Value vectors have an analogous role as in the attention mechanism.

In this architecture, all linear projection vectors involved are obtained by a linear interpolation of current and previous time steps tokens. For the time-mixing computation, we have:

$$r_t = W_r \cdot (\mu_r \odot x_t + (1 - \mu_r) \odot x_{t-1})$$

$$k_t = W_k \cdot (\mu_k \odot x_t + (1-\mu_k) \odot x_{t-1})$$
$$v_t = W_v \cdot (\mu_v \odot x_t + (1-\mu_v) \odot x_{t-1})$$

While for the channel-mixing we have:

$$r'_t = W'_r \cdot (\mu'_r \odot x_t + (1-\mu'_r) \odot x_{t-1})$$

$$k'_t = W'_k \cdot (\mu'_k \odot x_t + (1-\mu'_k) \odot x_{t-1})$$

The computation of the WKV operation gets inspiration from the attention-free transformer, but treats W as a channel-wise vector, and its update is formalised with the following equation:

$$\texttt{wkv}_t = \frac{\sum_{i=1}^{t-1} e^{-(t-1-i)w+k_i} \odot v_i + e^{u+k_t} \odot v_t}{\sum_{i=1}^{t-1} e^{-(t-1-i)w+k_i} + e^{u+k_t}}$$

To conclude, the output vector after the WKV is given by:

$$o_t = W_o \cdot (\sigma(r_t) \odot \texttt{wkv}_t)$$

while in the channel-mixing block we have a similar operation:

$$o'_t = \sigma(r'_t) \odot (W'_v \cdot \texttt{max}(k'_t, 0)^2)$$

Chapter 3

Explaining the Gap

In this section, I explain and replicate the findings shown in Zoology [7] and present its main drawbacks that inspired the experiments of my thesis.

3.1 Zoology and The Gap

So far I introduced two main categories of Deep Learning models: On one hand, transformers model (or attention models), which explicitly compute all the interactions between the tokens processed via the attention mechanism. On the other hand, a series of recurrent-gated convolution models, that instead update an hidden state h when processing the tokens in the sequence.

The latest are usually more efficient from a computational point of view (compared to the $O(L^2)$ bottleneck of the attention mechanism) and have incredible performance on the AR task (as show in [2], [3], [4]), even when the sequence length is $> 10,000$, where instead the transformer architecture usually encounters memory errors. However, when these models are compared in actual language modeling, the transformer architecture has a significant edge in terms of perplexity (i.e. [7])). So the question that inspired my thesis is:

"Why are recurrent model not capable of transferring their impressive performance in Associative Recall in actual Language Modeling?"

Fortunately, *Arora et al.* asked themselves the same question and tried to give an answer with their work **Zoology** [7], an in-depth comparison of attention and non-attention models.

In their work, they investigate the performance of these two categories of models in two specific scenarios: the first, in which they compute the

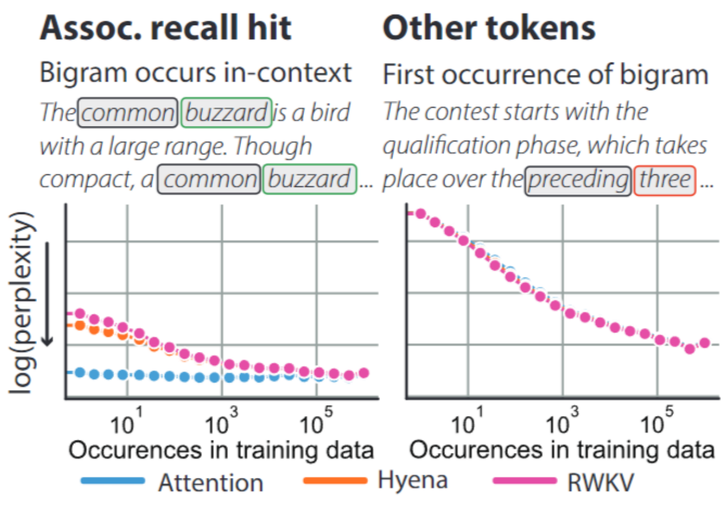

Figure 3.1: Gap in performances with and without Associative Recall hits. The graph is taken from the original work of Zoology[7].

perplexity with pieces of text similar to the one on the left of Figure 3.1. These are simply the key-value pairs of the associative recall task that I mentioned in Chapter 2, but with proper words from a vocabulary. In Zoology, the authors called these "**Associative Recall Hits**" (AR Hits). The second scenario (right of Figure 3.1) is instead a piece of text without AR hits, where all the words in the text are seen for the first time or simply are not correlated with others.

In Zoology, the authors noticed that while in the second case, the performances of all the models are comparable, in the first case it seems like the transformer architecture have a clear advantage. Apparently, when dealing with AR hits seen fewer times in the sequence, attention is capable of recalling tokens more effectively, while the recurrent models need multiple scans before doing the same with confidence. The author claims that this skill that the transformer has, is responsible for the overall gap in performance in language modeling with recurrent models by a factor of 82%. From the perspective of attention models, these results show that solving associative recall is essential to being a good LLM. However, it is still unclear why recurrent models are not capable of transferring their impressive performance on AR to proper language modeling.

From the point of view of the author of Zoology, the problem is the actual task of AR: from their perspective, the task is too easy and not a suitable proxy for language modeling. This is due to two main problems:

- **Vocabulary size**: In the associative recall task, the vocabulary size is in the order of 50 tokens. This does not really represent natural language, were there are thousands of words that could potentially be recalled from a vocabulary;
- **Number of key-value pair to be inferred**: In the AR, the model needs to infer just one value given a key, but again this does not reflect what happens in natural language, where instead multiple words recall multiple others.

3.2 Multi-Query Associative Recall

So far the authors claim that the AR task is not challenging enough, or in other words it is not a good proxy for natural language. So what models need to learn to solve this synthetic task, is not transferable to language modeling. To solve this problem, the author proposed a more challenging version of the AR task, the Multi-Query Associative Recall (MQAR) task, that should be a more suited proxy for language. More specifically here are the two main differences with AR:

- **Vocabulary size**: Now the vocabulary size is increased to almost $9,000$ tokens, making it extremely more challenging;
- **Number of key-value pair to be inferred**: Now given a sequence like

  $$A \;\; 6 \;\; I \;\; 9 \;\; C \;\; 7 \;\; P \;\; 1 \;\; S \;\; 4 \;\; D \;\; 2$$

  we ask the model to recall multiple tokens at the same time

  $$C \rightarrow \; ? \;\; A \rightarrow \; ? \;\; D \rightarrow \; ?$$

Given this new task, Zoology compares all the models in the MQAR task with different sequence lengths, relative number of key-value pairs to recall and different model dimensions. The models compared in this and the following experiments are the ones already described in the Chapter 2 (Attention, Hyena, H3 and RWKV) plus two ad-hoc models introduced by the authors of Zoology:

- **BaseConv**: A naive implementation a of recurrent-gated convolutional model. The idea is to use this model as a lower bound of the experiments. In fact, if we are able to improve the performance of this model, we expect to also increase the performance of other recurrent-gated models;
- **Based**: A recurrent model that tries to mimic the computations of the attention module.

3.3 Experimental Details

In this section, I present the experimental details that were used in the experiments replicated from Zoology and all the experiments conducted in this work. In case of changes relative to architecture, hyperparameters or similar, these will always be explicitly stated.

- **MQAR task**: Given the sequence of key-value pairs (context) and some keys already seen in the sequence, the models need to infer the relative values from the context. The models need to infer all the key-value pairs requested in order to consider the data point correct;
- **Graph explanation**: In every experiments there will be four subgraphs, one for each sequence length ($64, 128, 256$ and 512). Each subgraph will contain 7 models in total, 2 attention-models (Attention and Based) and 5 recurrent-models (Mamba, Hyena, H3, RWKV and BaseConv). On the y-axis there is accuracy, while on the x-axis there is the model dimension on log scale;
- **Optimizer and Schedule**: For each model, four learning rates in the log space $[10^{-4}, 10^{-2}]$ are swept. The optimizer used is AdamW with weight decay 0.1, warm-up duration 10% and linear warm-up;
- **Epochs and Batch size**: All the models are trained for 150 epochs. The batch size is 16 for sequence length ≥ 512, 32 for sequence length $= 256$ and 64 for sequence length ≤ 128;
- **Width and Depth**: Each of the models evaluated in the experiments use two layers. Each layer is composed by one sequence mixer (specific of the architecture), one Multilayer Perceptron and layer normalization in between. The model dimensions used are 64, 128, 256 and 512. The sequence length used and the corresponding key-value pair to infer are $(64; 4)$, $(128; 8)$, $(256; 16)$, and $(512; 64)$ respectively;
- **Position Information**: Position Embedding is used for all the attention-like models (Attention and Based), while it is not used for the other recurrent models;
- **Data**: Models are trained and evaluated on a train set and a test set with $100,000$ and $3,000$ data points respectively. I want to highlight that the tokens used as key and value are always chosen randomly for each sequence. In this way the models cannot learn a general key-value mapping, but need to learn how to recall the key-value pairs seen in context.

All the relative code can be found in the original work of Zoology [7] at https://github.com/HazyResearch/zoology/tree/main.

3.4 Replication and Drawbacks of Zoology

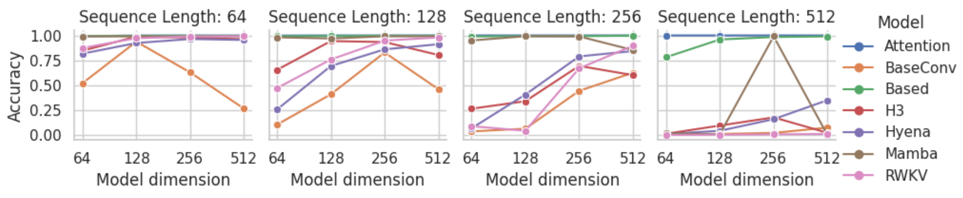

Figure 3.2: Zoology reproduction. There are four subgraphs and each one keeps a fixed sequence length $(64, 128, 256$ and $512)$. For each sequence length, 7 models are shown in total, 2 attention-models (Attention and Based) and 5 recurrent-models (Mamba, Hyena, H3, RWKV and BaseConv). On the y-axis there is accuracy, while on the x-axis there is the model dimension on log scale. Intuitively, we would expect bigger models to more easily solve the task.

In Figure 3.2, I show my replication of the Zoology experiments with the addition of Mamba, which was not inserted in the original code. I want to highlight that, intuitively, we would expect bigger models to more easily solve the task.

The results of the simulation show three different patterns based on the characteristics of the models involved:

- **Recurrent models**: These models show the behavior that we would have expected. Recurrent models easily solve the task with shorter sequences, while as soon as the sequence length increases, the task is solvable only when the model dimension matches or surpasses the sequence length. However, the hardest setting with a sequence length of 512 seems almost impossible to solve for all the models of this kind;
- **Attention models**: On the opposite side, Attention and Based are capable of always solving the task with perfect scores, regardless of the sequence length and the model dimension. It seems like these architectures are inherently capable of solving this specific MQAR task;
- **Mamba**: The selective SSM has a sort of hybrid behavior. When the sequence length is smaller (≤ 256), Mamba has the same performances of attention models, while with longer sequences it has instead a wiggling behavior (see as an example how with sequence length of 512 the task is solvable with a model dimension of 256 but not with a model dimension of 512).

These results follow the same gap in performance between attentions and recurrent models in language modeling: apparently, the latter are not capable of solving the MQAR tasks.

However, in this work I investigates two main problems related to the experiments proposed by Zoology:

- **Lack of Optimization**: The author did a poor grid search when tuning the learning rates (just 4 values between 10^{-4} and 10^{-2} which are not properly spaced). This represent a critical limitation since optimization is one of the main challenges when dealing with recurrent models and can make a significant difference in what task such models are capable of solving. This insight is noticeable in the case of Mamba with a sequence length of 512 at the right of Figure 1. In fact, the model is capable of solving the task with a dimension of 256 but not with a dimension of 512 (that should be more powerful and solve the task more easily);
- **Arbitrary use of 2 layers**: The author always stacked two blocks when creating the models. This choice comes from the discovery of induction heads ([10]), a particular behavior of transformers with at least 2 layers of attention that allows a significant gain in In-Context Learning (ICL) capabilities. However, why this choice should be applied also in such a setting is not clear.

Chapter 4

Experiments

In the following section, I investigate how these two factors (optimization and scaling) influence the performance of models and what improvements are needed to solve the MQAR task and generally to better solve language modeling tasks.

4.1 The Role of Optimization

So far, one of the main problems of the experiments conducted in Zoology is the lack of proper optimization, especially when looking at the grid search for tuning the learning rate (lr) of the models. The author chose a too-narrow range (between 10^{-4} and 10^{-2}), too few values (just 4) and also values not properly spaced in the range. This can be extremely harmful when dealing with recurrent models, where the main challenge is to find a suitable lr, and it can make a significant difference in which tasks such models are capable of solving or not.

In this work, I proposes a better grid search for the learning rate to properly tune each model that will also be used for all the experiments in this work:

- The range is set to be $[0.3, 0.00001]$;
- The values chosen are $0.3, 0.1, 0.03, 0.01....0.00003, 0.00001$. The good property of these values is that given any value, the previous and next one differ with a factor of 3, making them equally scaled on the logarithmic scale;
- After trying all the values of the previous step, the best value lr^* is selected;
- Now a new more-fined grid search $[\frac{lr^*}{3}, \frac{lr^*}{2}, lr^*, 2lr^*, 3lr^*]$ is set. All the values are tried and if the new best lr is not the same as lr^*, a new grid

search is made. This is done until the new best lr is equal to lr^*. Other than confirming that lr^* is the best lr, this process also ensures that lr^* is not a boundary value (i.e. if we slightly increase or decrease its value the training process does not overshoot);

- Finally, lr^* is used with 3 different seeds and the results show the average performances. In this case the used seeds are 123, 777 and 42.

Table 4.1 shows a comparison of the starting grid search used for choosing the learning rates of the models. Figure 4.1 shows the results with a more

Authors	**Values**
Zoology	[0.01, 0.00214, 0.00047, 0.0001]
Me	**[0.3, 0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001, 0.00003, 0.00001]**

Table 4.1: Learning rate grid search comparison.

optimized learning rate tuning, with also two new size of models, 1,024 and 2,048, to have an even further investigation.

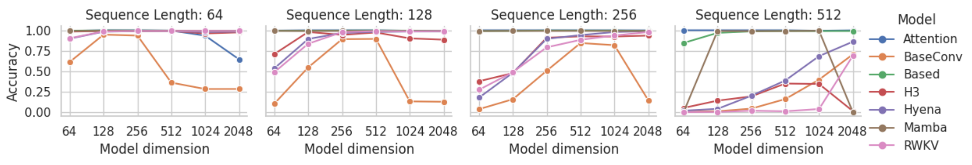

Figure 4.1: Zoology reproduction with better learning rate tuning.

The results show a completely different behavior: now for a sequence length ≤ 256, recurrent models gets $\approx 100\%$ accuracy, while even with a sequence length of 512 these models achieve better results than before, especially with Hyena. This significant improvement obtained by just choosing the learning rate more carefully is deeply connected with the different nature of transformer and recurrent models. In the first case, given a sequence of tokens $(x_1, ...x_n)$, the attention mechanism can compute all the interactions between each token in the sequence explicitly through the matrices W_Q, W_K and W_V, assuming that the model has enough memory to do so. Instead, recurrent models process information differently by updating their hidden state h_t with the information of each token processed implicitly. In other words, the model needs to efficiently compress the current x_t information into a hidden representation h_t and then retrieve the information needed to make inference y_t. Intuitively, the learning rate defines the size of the steps for this compression-expansion process and without a proper choice of this hyperparameter, recurrent models cannot properly leverage their fundamental component.

4.2 The Role of Depth and Width

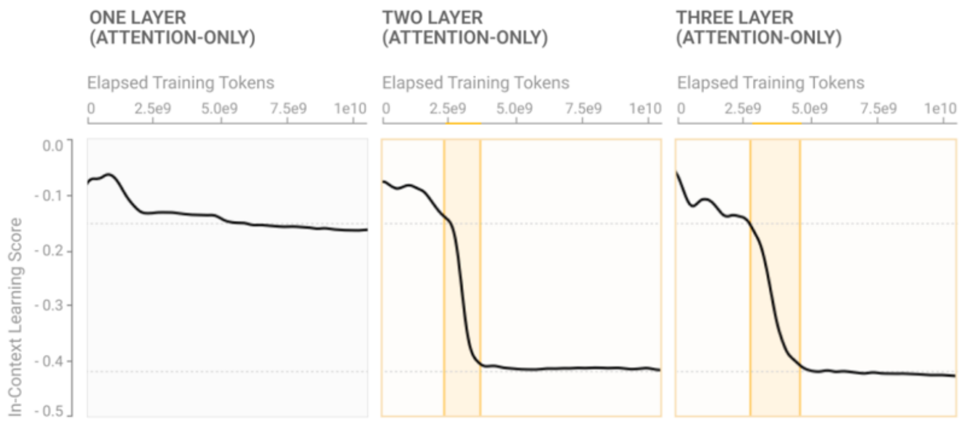

Figure 4.2: In-Context Learning score of 1,2 and 3-layers transformers models. The graph is taken from the original work of In-Context Learning and Induction Heads [10].

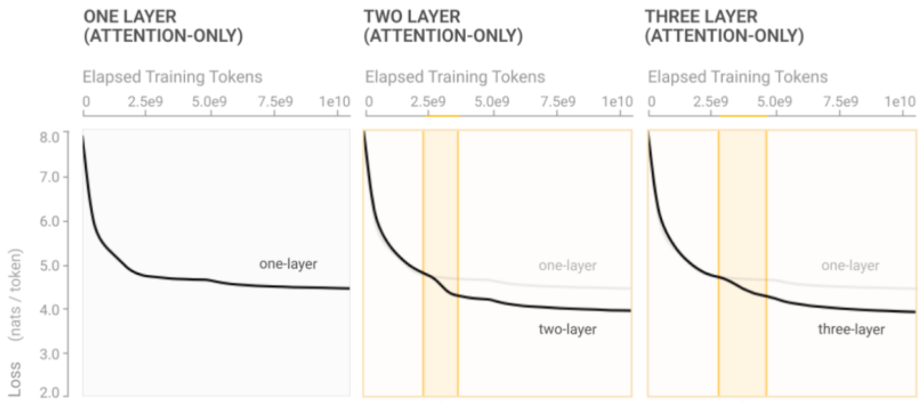

Figure 4.3: Training Loss of 1,2 and 3-layers transformers models. The graph is taken from the original work of In-Context Learning and Induction Heads [10].

Another problem of the experiments conducted in Zoology is the choice of the number of layers used to build the models. The author used 2 blocks for each architecture, without properly explaining why and how this choice affects performances in models. For instance, it would be more reasonable to make experiments with just one block, in order to properly understand which are the fundamental differences between attention and recurrent models.

The idea of interleaving 2 blocks when building a transformer comes from the literature related to Associative Recall and generally In-Context Learning (ICL), in particular from the discovery of *"Induction Head"* in [10], where the authors were conducting experiments on the capabilities of transformer in ICL tasks such as in few-shot learning. The main insight from this work was that during training, with transformers with at least 2 layers, a special kind of attention heads called *"Induction heads"* is formed, giving a sudden boost in performances in terms of:

- **In-Context Learning scores**: An ad-hoc score that computes the difference in loss between the 50*th* and the 500*th* tokens in a 512-length sequence) as shown in Figure 4.2;
- **Training loss**: As shown in Figure 4.3.

More formally, induction heads are implemented by a circuit consisting of a pair of attention heads in different layers that work together to copy or complete patterns. The first attention head copies information from the previous token into each token, making it possible for the second attention head to attend to tokens based on what happened before them, rather than their own content. Specifically, the second head (the proper "induction head") searches for a previous place in the sequence where the present token **A** occurred and attends to the next token (call it **B**), copying it and causing the model to be more likely to output **B** as the next token. That is, the two heads working together cause the sequence ...[**A**][**B**]...[**A**] to be more likely completed with [**B**].

Induction heads are named by analogy to inductive reasoning, where we might infer that if **A** is followed by **B** earlier in the context, **A** is more likely to be followed by **B** again later in the same context. Induction heads are capable of crystallizing that inference. They search the context for previous instances of the present token, attend to the token which would come next in the pattern repeated, and increase its probability in terms of logit. Induction heads attend to tokens that would be predicted by basic induction (over the context, rather than over the training data).

Notice that induction heads are implementing a simple algorithm, and are not memorizing a fixed table of n-gram statistics. The rule [**A**][**B**] ... [**A**] $\rightarrow$ [**B**] applies regardless of what **A** and **B** are. In fact, the rule [**A**] $\rightarrow$ [**B**] could be used to understand the rule [**A'**] $\rightarrow$ [**B'**], where [**A**] $\approx$ [**A'**] and [**B**] $\approx$ [**B'**]. This means that induction heads can in some sense work out-of- distribution (OOD), as long as local statistics early in the context are representative of statistics later. This hints that they may be capable of more general and abstract behavior.

However, it is not clear why this choice, which was made on language modeling task, should also be applied in the simpler task of MQAR. So, to better understand the role of layers not only in attention models but also in recurrent models, in this work I investigate what happens when models with just 1 layer are evaluated in the MQAR task. The results are shown in Figure 4.4.

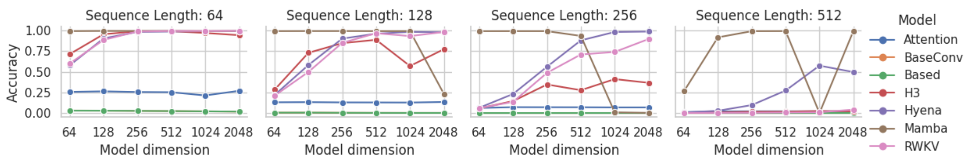

Figure 4.4: Results with just 1-layer models.

The results show something surprising and completely different from what was seen before:

- On the one hand, attention models are not capable anymore of solving the task and their general performance decreases as the sequence length increases. This behavior however is different from the case of recurrent models, because, when fixing the sequence length, attention models have the same performances regardless of the dimension of the models, whereas intuitively we would expect the bigger model to have some sort of advantage;
- On the other hand, recurrent models have roughly the same results as the 2-layer experiments, solving the task with a sequence length ≤ 256 but struggling with 512. However, when fixing the sequence length, these models improve performances by increasing their dimension.

It seems like transformer models benefit the most from scaling in terms of depth, as if there is a connection between the number of layers of attention and the type of tasks solvable for the model, while recurrent models benefit the most from scaling in terms of width.

I hypothesize that this is related to the different nature of the two models: attention models try to explicitly compute all the interactions between each token and to do so the architecture needs more blocks to mix the information across tokens. Whereas recurrent models need to efficiently compress information in their hidden state and its dimension (or in other word, its space) is related to the model dimension. So, by increasing the width of the model, there is a bigger hidden representation state, allowing for easier compressions.

4.3 Learning Curves

After the results obtained so far, in this work I also investigate learning curves more in depth, in order to better understand the differences in the learning dynamic of the models presented to highlight their differences and similarities. All the following experiments consider cross-entropy as loss function.

- **Attention models with 2-layer**: In this case, the models are always capable of solving the MAQR task, achieving perfect accuracy regardless of the sequence length. Figure 4.5 shows the loss dynamic typical of attention models in this setting, where after an initial phase of constant error, there is a significant drop, allowing the models to easily solve the task after few epochs. As expected, this follows the phenomenon of the induction head mentioned before, allowing attention models to increase their performances in ICL tasks;

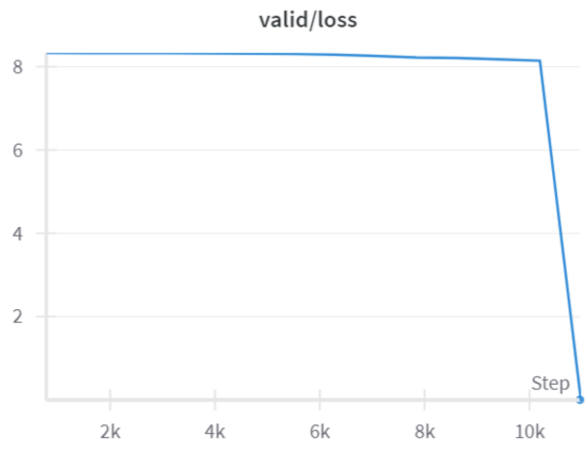

Figure 4.5: Loss curve of 2-layer attention models.

- **Attention models with 1-layer**: This is the case in which the models are not capable of solving the MQAR task, with a decrease in performance as the sequence length increases. In addition, when fixing the sequence length, performances remain constant regardless of the dimension of the model, whereas instead an increase in performance with bigger models would be expected. What is interesting in this case, is the typical behavior of the loss in this setting shown in Figure 4.6, that even if too high to solve the task, still shows a big drop. It seems that the attention model with 1-layer still tries to build something similar to induction heads, but needs more blocks (or in other words, more complex representations) to leverage them in ICL tasks;

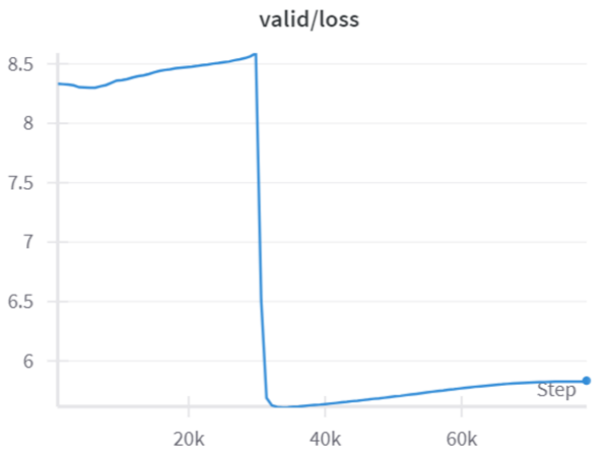

Figure 4.6: Loss curve of 1-layer attention models.

- **Recurrent models**: In this case, the models are capable of solving the MQAR task just if properly optimized. Figure 4.7 shows the expected unstable learning behavior when the model learning rate is not well-tuned. On the other hand, optimizing recurrent models gives the smooth learning curve shown in Figure 4.8, where at each time step the model becomes more capable of properly compressing information in its hidden state. In particular, this becomes easier when the dimension of the models is increased regardless of the number of layers used to build the model, suggesting how recurrent models capabilities depends on their efficiency in compress information rather than the complexity of the recurrence. These two behaviors appears in the same way regardless of the number of layers used to build the recurrent model.

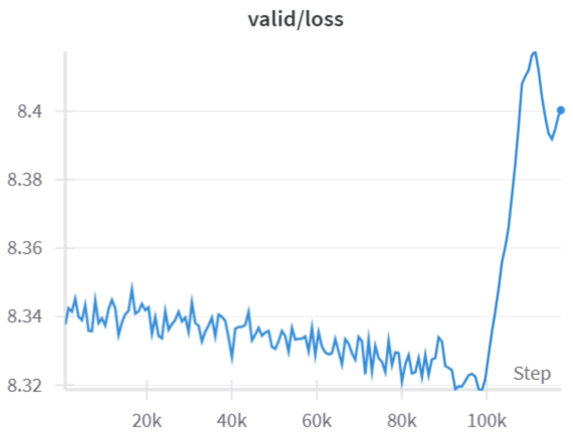

Figure 4.7: Loss curve of non-optimized recurrent models.

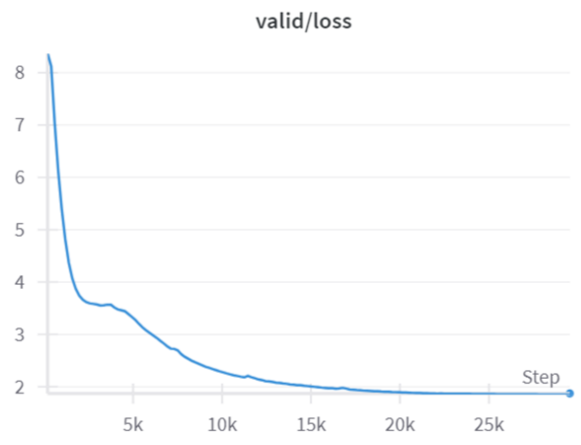

Figure 4.8: Loss curve of optimized recurrent models.

4.4 Vocabulary Extension

To investigate even further the capabilities of attention and recurrent models in the 1-layer setting, in this section I show one more last experiment on the influence of vocabulary size on performance.

Previous works ([18], [19]) showed that the transformer architecture benefits from increasing the vocabulary size in terms of perplexity and general performance. The goal of this experiment is to see if 1-layer attention models can leverage augmentation that usually benefits transformer architectures. In particular, in this experiment, I quadrupled the original vocabulary size of the MQAR task (from $8,192$ to $32,768$) and also increased accordingly the train set and test set size (respectively from $100,000$ to $400,000$ data points and from $3,000$ to $12,000$ data points). All the others hyperparameters remain unaltered. Results are shown in Figure 4.9.

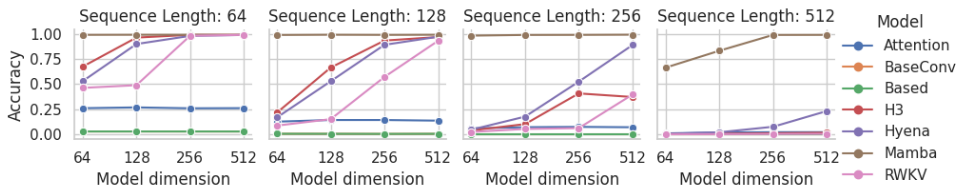

Figure 4.9: Result with increased vocabulary size.

The results are almost identical to the one with the original vocabulary size, both for attention and recurrent models, apart from RWKV, which interestingly is also the recurrent model most similar to attention, that got a general decrease instead. So what is clear is that attention models with 1 layer are inherently different from attention models with 2 or more layers, suggesting how scaling in depth has a crucial role in their capabilities.

4.5 Limitations

Even with the improvements proposed and the insights obtained, this part of my work still has some limitations that could inspire future research:

- **Optimization**: Even with the learning rate grid search proposed, getting the right value can still be challenging (i.e. Mamba is still unstable when scaled to bigger dimensions). In the experiments proposed in my work, the grid search was made from scratch every time and for all the model dimensions and sequence lengths. It would be more useful to find some correlation between learning rate, sequence length and model dimension to more easily transfer suited hyperparameters with different model settings;
- **Scaling**: All the experiments were made with models with at most 50 M parameters and with at most $2,048$ as model dimension and with a maximum sequence length of 512. However, it would be interesting to see how performance and behavior change when dealing with models with billions of parameters and sequence length in the range of thousands of tokens, like in DNA sequencing.

Chapter 5

Recurrent models applied to 3D Vision

So far I gave some insights into the differences between attention and recurrent models, suggesting how the setting around these models should be different in order to leverage their fundamental components. In this section, I show an application of such insights in the field of 3D computer vision, in particular on how to reconcile a recurrent model (Mamba) with data modalities that are inherently sets. Parts of the work described in this chapter were originally presented in [20].

5.1 Deep Learning in 3D Vision

The transformer architecture has become the dominant technology for powering large-scale deep learning systems. Since their introduction by [1], transformers have seen widespread adoption across text [21, 22], image [23, 24, 25], and video [26, 27, 28, 29] domains, as well as in multimodal applications [30, 31]. In 3D vision, particularly for point cloud analysis, transformers achieve state-of-the-art performance [32, 33, 34, 35], frequently outperforming convolution-based methods [36, 37] at scale.

While the architecture of transformers is well-suited for modern hardware, their softmax attention mechanism significantly impacts model complexity, scaling quadratically with sequence length in text or the number of patches in image, video, or point cloud data. This quadratic scaling has driven extensive research into alternative sequence mixer strategies over the years, including approaches like separable attention [38, 39, 40, 41, 42, 43, 44] and optimized GPU implementations of softmax attention [17, 45, 46]. However, a major breakthrough in addressing this issue came recently with the emergence of state-space models (SSMs) like Mamba [2] and other parallelizable token mixers [6, 47, 48, 49, 50, 51].

SSMs are RNN-like, highly parallelizable sequential blocks that trace back to foundational work [14, 3], offering linear complexity with respect to sequence length. This efficiency enables them to handle long-context scenarios effectively, which is crucial for tasks such as audio processing [52] and DNA modeling [53]. Beyond their efficiency, models like Mamba, xLSTMs, and other RNN/linear attention variants often demonstrate improved downstream performance [54] and enhanced reasoning capabilities in tasks such as the long-range arena [12] and other challenging text benchmarks where transformers may falter [51].

Following their success in text and audio, Mamba and similar models have been extended to 2D vision [55, 56, 57] and various 3D data applications [58, 59]. Notably, they show promise in processing point clouds [60, 61, 62], which often involve datasets like ScanNet [63] containing over 100k points. In 1D domains such as text or audio, Mamba processes data sequentially (left-to-right or bidirectionally) without requiring positional embeddings [54]. However, applying Mamba to 2D and 3D data introduces unique challenges. Unlike sequential 1D data, 2D and 3D datasets are inherently unordered, and positional information must be explicitly incorporated.

In non-causal attention-based models, these operations are treated as set operations [64], with positional information encoded directly in features [1, 65] or attention matrices [66]. For example, BERT-like encoders [67] without positional embeddings reduce to bag-of-words models, while causal self-attention can recover positional information in deeper layers. Thus, although attention mechanisms conceptually unify 1D, 2D, and 3D data by treating them as sets with added positional information, the introduction of order-sensitive sequential blocks like Mamba raises conceptual challenges. This motivates my investigation, particularly in the context of 3D data processing.

How should a sequential model be applied to non-sequential data, e.g. a point cloud?

Exploring this question is scientifically fascinating, highly relevant, and essential for unlocking the full potential of emerging efficient attention mechanisms in the 3D domain. Upon reviewing the rapidly expanding body of work on Mamba applications in 3D vision, two recurring patterns emerge[1].

(A) 3D point cloud data has to be converted into an *ordered* sequence before Mamba can be applied. This has been achieved with different strategies such as reordering the points along axis and replicating the sequence [60, 61] or scanning it from different directions [62].

[1]Similar discussion would hold for 2D data, see e.g. [55].

(B) Similar to transformers, positional embeddings are employed. However, this information is conceptually redundant because (1) the features themselves inherently contain positional information, and (2) Mamba implicitly utilizes the patch ordering within the constructed sequence. It is worth noting that in text applications, Mamba is often used without positional embeddings [54].

Although Mamba's performance on 3D data is already promising—frequently exceeding transformers in both accuracy and processing speed—points A and B above highlight significant challenges when applying Mamba to point clouds. These challenges could impact its robustness and ability to generalize to out-of-distribution data. To deepen our understanding and refine optimal preprocessing strategies for Mamba-powered models in 3D applications, I present the following contributions:

1. I draw attention to the problem of sequence construction when applying Mamba to 2D or 3D data. I complement the discussion with both theoretical considerations on invariances and positional embeddings (Sec. 5.4.3) and ablations (Sec. 5.5).

2. I introduce NIMBA[2], a Mamba-like model that feeds 3D data points based on an intuitive 3D-to-1D reordering strategy that preserves the spatial distance between points (Sec. 5.4.3). This strategy allows for *safe removal of positional embeddings* without affecting (most times, improving) performance. This is in stark contrast to all previously introduced Mamba strategies in point clouds where my ablation reveal a performance drop when positional emebddings are not used. Along with improved efficiency, my results (Sec. 5.5) showcase how principled ordering along a point cloud can improve performance of Mamba models in this setting.

3. I show how our ordering strategy in NIMBA drastically improves robustness of the model against data transformations such as rotations and jittering (Sec. 5.5).

I compare my contributions with previous work in Table 5.1.

5.2 Related Work in 3D Vision

Point Cloud Transformers. Transformers, originally developed for natural language processing (NLP), have proven highly effective in point cloud analysis due to their global attention mechanisms and inherent permutation invariance. Early models like Vision Transformer (ViT) [23] showcased that

[2]The name NIMBA is derived from the combination of `Nimbus` (latin for "dark cloud") and `Mamba`.

Model	Backbone	Seq len	Bidirectional	Pos emb
PCT	Transformer	N	×	✓
PointMAE	Transformer	N	×	✓
PointMamba	Mamba	$3N$	×	✓
Point Cloud Mamba	Mamba	$3N$	✓	✓
OctreeMamba	Mamba	N	✓	✓
Point Tramba	Hybrid	N	✓	✓
PointABM	Hybrid	N	✓	✓
NIMBA (Ours)	Mamba	N	×	×

Table 5.1: Comparison of Models based on Architecture, Sequence Length, Directionality and Positional Embedding. I denote with N number of points in the point cloud. The Table is taken from the original work of NIMBA [20].

transformers could surpass convolutional neural networks (CNNs) in classification tasks by directly applying attention to image patches. This success paved the way for their adoption in point cloud processing, where modeling global features is essential.

Point-BERT [33] adapted BERT's masked modeling approach to 3D data by employing a discrete tokenizer to convert point patches into tokens and using self-supervised pre-training to reconstruct masked points. Building on this, Point-MAE [34] introduced masked autoencoding to learn latent representations by reconstructing missing regions from masked inputs. These methods leveraged large unlabeled datasets and achieved superior performance compared to traditional models, though the quadratic complexity of self-attention posed scalability challenges.

OctFormer [68] tackled this limitation by incorporating octree-based attention, reducing computational overhead through local window partitioning while maintaining strong performance on large-scale tasks. PointGPT [69] adopted an autoregressive framework inspired by GPT, treating point patches as sequential data to predict subsequent patches. This pre-training approach demonstrated robust generalization in few-shot and downstream tasks, further showcasing transformers' adaptability for point cloud processing. Similarly, PCT [32] utilized a transformer architecture designed with permutation invariance to process unordered point sequences. By integrating farthest point sampling and nearest neighbor search, PCT effectively captured local context, achieving state-of-the-art results in tasks such as shape classification and part segmentation.

Point Cloud State Space Models. The application of state space models (SSMs) to point cloud analysis has recently garnered attention as a promising solution to the computational challenges faced by transformer-based architectures. While transformers excel at capturing global dependencies, their quadratic complexity hinders their scalability for high-resolution point clouds. In contrast, SSMs, such as the Mamba architecture, provide linear complexity and efficient long-range modeling. However, a significant challenge in applying SSMs to point clouds lies in the unordered nature of the data, which conflicts with the sequential processing requirements of SSMs. To address this, researchers have developed methods to transform point clouds into sequences.

A prevalent strategy in recent research involves designing ordering methods that preserve spatial relationships during sequence conversion. For instance, PointMamba [60] and Point Cloud Mamba (PCM)[61] employ axis-wise reordering techniques and sequence replication to enhance SSMs' ability to capture both local and global structures. Other approaches use hierarchical data structures to reflect spatial hierarchies. OctreeMamba[62], for example, adopts an octree-based ordering scheme, organizing points in a z-order sequence to preserve spatial relationships while capturing multi-scale features. While maintaining spatial relationships during serialization is vital, improving local feature extraction within SSMs is equally crucial for point cloud analysis. Although SSMs efficiently model long-range dependencies, capturing fine-grained local details remains a challenge. Mamba3D [70] addresses this by introducing a Local Norm Pooling block to enhance local geometric representation and employs a bidirectional SSM to model both tokens and feature channels, balancing local and global structure representation without increasing computational complexity.

Building on work such as [54, **?**], another research direction combines the strengths of Transformers and SSMs to achieve improved performance and efficiency. PoinTramba [71], for example, uses Transformers to capture detailed dependencies within point groups, while Mamba models relationships between groups using a bidirectional importance-aware ordering strategy. By reordering group embeddings based on importance scores, this method mitigates random ordering issues in SSMs and enhances performance. SSMs have also found applications in point cloud completion and filtering. 3DMambaIPF [72] leverages Mamba's selection mechanism with HyperPoint modules to reconstruct point clouds from incomplete inputs, preserving local details that Transformers often miss. For filtering, it combines SSMs with differentiable rendering to reduce noise in large-scale point clouds, improving alignment with real-world structures and efficiently handling datasets with hundreds of thousands of points—scenarios where other methods often falter.

These advancements underscore the effectiveness of SSMs in addressing critical challenges in point cloud analysis, offering efficient and scalable solutions for various tasks. With ongoing innovations in serialization, feature extraction, and hybrid architectures, SSMs are proving to be a valuable tool in advancing 3D vision applications. In this work, I aim to further build on these results by presenting a simple, robust, and principled solution for constructing input sequences tailored for Mamba-like models.

5.3 Datasets

In my experiments, I evaluate the performance of my model using three publicly available 3D datasets: ModelNet40, ScanObjectNN and ShapeNetPart.

ModelNet40 [73]: It is a widely used benchmark synthetic dataset used to evaluate 3D object classification models. It consists of 12,311 CAD models across 40 categories, representing clean and noise-free 3D shapes, such as airplanes, chairs, and cars, offering a diverse set of 3D shapes.

ScanObjectNN [74]: It presents a more challenging real-world scenario by offering around 15,000 objects across 15 categories, scanned from real-world indoor scenes. Unlike the controlled environment of CAD datasets, the objects in ScanObjectNN are captured in cluttered and noisy environments, introducing additional complexity due to background noise, occlusion, and deformation. The diversity of real-world settings in this dataset makes it particularly suited for evaluating the robustness of models in practical object classification tasks. The dataset comes in three variants with different degree of difficulty:

- OBJ-ONLY: the easiest variant in which there is only the object in the scene. This is the most similar variant to a CAD analogous.
- OBJ-BG: an intermediate variant in which there is also the background. I focused mostly on this variant since it is the most similar to the real world.
- PB-T50-RS: the hardest version that adds some perturbations to the objects and can be used as a benchmark to test the robustness on the classification task

ShapeNetPart [75] It is a widely recognized benchmark for 3D shape segmentation tasks. This dataset is a subset of the larger ShapeNet repository and includes 31,693 3D CAD models categorized into 16 common object classes such as chairs, planes, and tables. Each model is richly annotated with detailed geometric and semantic labels, providing valuable information for training and evaluating segmentation algorithms.

5.4 Model Design

I start in Sec. 5.4.1 by overviewing the processing strategies common in the point cloud literature. In Sec. 5.4.2 I recall the basic properties of Mamba and self attention, highlighting their connections. I continue in Sec. 5.4.3 by describing how Mamba-like processing of point cloud data leads to interesting considerations around the effects of assigning an order to patches in 3D space. Then I analyze the PointMamba strategy in Sec. 5.4.3 and in Sec 5.4.3 I describe the proposed methodology.

5.4.1 Basic Strategies in Point Cloud Analysis

I outline the typical pipeline used for point cloud analysis in recent deep models. These are not specific to my model, but will allow me to make connections and simplify the discussion.

Preprocessing. The objective of the preprocessing stage is to reduce the number of points in the point cloud while maintaining the data's structural integrity, enabling more efficient computations in later steps. Formally, let $\mathcal{P} = \{\mathbf{p}_i \mid \mathbf{p}_i \in \mathbb{R}^3, i = 1, \ldots, N\}$ represent the point cloud, where N is the total number of points, and $\mathbf{p}_i = (x_i, y_i, z_i)$ denotes the 3D coordinates of each point. Following normalization, the process typically involves two steps:

1. *Center Selection:* A total of n_c points are chosen using the Farthest Point Sampling (FPS) algorithm. FPS works by iteratively selecting points that are maximally distant from one another, ensuring the sample effectively represents the original point cloud. These selected points, referred to as "centers" $\{C_i\}_{i=1}^{n_c}$, capture *global* information about the object.
2. *Patch Creation:* For each center, the n_p closest points are selected using the k-Nearest Neighbors (kNN) algorithm. This process generates a set of patches $\{P_i\}_{i=1}^{n_c}$, with each patch centered around one of the selected centers, thereby capturing more localized information about the object.

The values of n_c and n_p are hyperparameters of this preprocessing stage.

Patch Embedding. Each patch P_i is transformed into a fixed-dimensional vector $\mathbf{p}_i$ via a series of expansions, convolutions, and linear projections. This embedding process represents a pointwise transformation from $\mathbb{R}^{\mathrm{BS} \times n_p \times 3}$ to $\mathbb{R}^{\mathrm{BS} \times n_p \times d_e}$. In this context, BS denotes the batch size, n_p refers to the number of points per patch, and d_e represents the embedding dimension. The embedding effectively captures the local geometric details within each patch, which are essential for comprehending the finer aspects of the object's structure.

Center Embedding. Each center C_i is mapped to a fixed-dimensional vector $\mathbf{c}_i$ to encode global *positional information* and offer context for the interactions between various patches. This embedding process constitutes a pointwise transformation from $\mathbb{R}^{\mathrm{BS} \times n_c \times 3}$ to $\mathbb{R}^{\mathrm{BS} \times n_c \times d_e}$.

In line with the approach used in transformer-based models [1], the center embedding plays a role similar to that of positional embeddings in point cloud analysis. By emphasizing the capture of spatial relationships throughout the entire point cloud, it is thought to offer a complementary perspective to the local information captured by patch embeddings.

5.4.2 Attention and Mamba in 3D Vision

In this subsection, I want to recall some key aspects of attention and recurrent models that plays a major role in 3D vision modeling.

To begin, I represent a generic input as $X \in \mathbb{R}^{N \times d}$, consisting of N elements in d dimensions. In the context of Sec. 5.4.1, X refers to the sequence of patch embeddings, potentially augmented with positional embeddings. I use X_i to denote the i-th row of X, which corresponds to an input token in text or a patch/point cluster in vision. I then describe the attention and Mamba-like processing applied to X, resulting in updated representations $Y \in \mathbb{R}^{N \times d}$.

Attention. The standard self-attention block [1] consists of three matrices: W_Q, W_K, and W_V, which are the learnt parameters of the model. These matrices, when multiplied with the input $X \in \mathbb{R}^{N \times d}$, yield the queries $Q \in \mathbb{R}^{N \times d}$, keys $K \in \mathbb{R}^{N \times d}$, and values $V \in \mathbb{R}^{N \times d}$: $Q = XW_Q$, $K = XW_K$, $V = XW_V$. These are combined to produce the output $Y \in \mathbb{R}^{N \times d}$.

$$Y = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right) V, \tag{5.1}$$

where softmax is applied row-wise. Assuming for simplicity W_V is the identity matrix, we get

$$Y = \Phi^X_{\text{SDPA}} \cdot X, \tag{5.2}$$

where $\Phi_{\text{SDPA}} \in \mathbb{R}^{N \times N}$ mixes tokens as follows:

$$\Phi^X_{\text{SDPA}} = \text{softmax}\begin{pmatrix} \frac{1}{\sqrt{d}} X_0 W_Q W_K^\top X_0^\top & \frac{1}{\sqrt{d}} X_0 W_Q W_K^\top X_1^\top & \cdots & \frac{1}{\sqrt{d}} X_0 W_Q W_K^\top X_N^\top \\ \frac{1}{\sqrt{d}} X_1 W_Q W_K^\top X_0^\top & \frac{1}{\sqrt{d}} X_1 W_Q W_K^\top X_1^\top & \cdots & \frac{1}{\sqrt{d}} X_1 W_Q W_K^\top X_N^\top \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{\sqrt{d}} X_N W_Q W_K^\top X_0^\top & \frac{1}{\sqrt{d}} X_N W_Q W_K^\top X_1^\top & \cdots & \frac{1}{\sqrt{d}} X_N W_Q W_K^\top X_N^\top \end{pmatrix}. \tag{5.3}$$

In causal self-attention, used e.g. in language modeling, the upper triangular portion of Φ_{SDPA} is set to 0. For vision application, Φ_{SDPA} is often used without masking.

Mamba. Architectures based on state-space models (SSMs) [3, 2, ?] compute the output Y through a dynamic recurrence of input signals. X is seen as a time-series where time flows from left to right: $X_1, X_2, \dots, X_N$. Starting from $Z_{i-1} = 0 \in \mathbb{R}^n$

$$Z_i = A_i Z_{i-1} + B_i X_i \tag{5.4a}$$

$$Y_i = C_i Z_i + D_i X_i, \tag{5.4b}$$

where Z_i is the hidden state of the system, and the dynamic matrices of appropriate dimensions A_i, B_i, C_i, D_i are functions of the model parameters as well as the input. The S6 block [2, **?**] parametrizes the recurrence as

$$A_i = e^{-\Delta_i W_A}, \qquad B_i = \Delta_i W_B X_i, \qquad C_i = W_C X_i, \qquad D_i = W_D X_i \tag{5.5}$$

and $\Delta_i = \text{softplus}(W_\Delta X_i + b_\Delta)$, with W_Δ, W_A, W_B, W_C, W_D are learnt *matrices* of appropriate dimensions, and b_Δ is a learnt bias. It is well known [15, 76, **?**, 77] that this system can be cast into an attention matrix representation [3], also known in the SSM literature as *convolutional* representation [78]:

$$Y = \Phi^X_{\mathrm{S6}} \cdot X, \tag{5.6}$$

where

$$\Phi^X_{\mathrm{S6}} = \begin{pmatrix} C_0 B_0 + D_0 & 0 & \cdots & 0 \\ C_1 A_1 B_0 & C_1 B_1 + D_1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ C_N \prod_{k=1}^{N} A_k B_0 & \cdots & C_N A_N B_{N-1} & C_N B_N + D_N \end{pmatrix}. \tag{5.7}$$

Architecture. Attention and S6 layers are commonly employed in deep networks, interspersed with MLPs, normalization components, and skip connections. In this work, I adopt the backbone of the Mamba architecture [2] and refer the reader to the original paper for further details, as well as to the appendix **??**.

5.4.3 Positional Embeddings and arrow of time in Mamba and Attention

There are two crucial macroscopic differences between Φ_{SDPA} and Φ_{S6}:

- Φ_{S6} is lower triangular, while Φ_{SDPA} is not.
- Φ_{SDPA} has an *isotropic* structure: entries close to the diagonal are computed similarly to entries far from the diagonal. Instead, in Φ_{S6} the distance to the diagonal affects computation: it affects in the number of A_is multiplied together in the formula for each entry.

This divergence between Mamba and Softmax attention is quite deep, and implications are strictly related to the two propositions below:

Proposition 5.1 (Softmax Attention) *$Y = \Phi^X_{SDPA} \cdot X$ is invariant to row-wise permutations Π of the input. For all X, Π and model parameters, we have $\Phi^{\Pi(X)}_{SDPA} \cdot \Pi(X) = \Pi(\Phi^X_{SDPA} \cdot X)$.*

[3] In modern variants of Mamba such as Mamba2 [**?**], the hidden dimension of Z in Eq. 5.4 is such that $\Phi_{\mathrm{S6}} \in \mathbb{R}^{N \times N}$. For earlier variants, the transformation is conceptually similar but has to be written in a slightly different form.

Proposition 5.2 (Mamba) *$Y = \Phi_{S6}^{X} \cdot X$ is* ***not invariant*** *to row-wise permutations Π of the input: there exists X, Π and model parameters such that $\Phi_{S6}^{\Pi(X)} \cdot \Pi(X) \neq \Pi(\Phi_{S6} \cdot X)$.*

Proposition 5.1 directly follows from the fact that attention is a *set operation* [1], and proposition 5.2 is also easy to prove:

The general formula describing S6 computation is $Y_k = C_k \sum_{j=0}^{k} (\prod_{k=j+1}^{k} A_k) B_k X_j$. Let us pick $N = 2$, we have $Y_0 = C_0 B_0 X_0$ and $Y_1 = C_1 A_1 B_0 X_0 + C_1 B_1 X_1$. Let Π swap the first and second inputs. For the reversed sequence, we have $\hat{Y}_0 = \hat{C}_0 \hat{B}_0 X_1$ and $\hat{Y}_1 = \hat{C}_1 \hat{A}_1 \hat{B}_0 X_1 + \hat{C}_1 \hat{B}_1 X_0$. We have to prove that for any realized value of $A, B, C, \hat{A}, \hat{B}, \hat{C}$, there exists a sequence X such that $Y_1 \neq \hat{Y}_0$, i.e. $C_1 A_1 B_0 X_0 + C_1 B_1 X_1 \neq \hat{C}_0 \hat{B}_0 X_1$. It is clear that converse would imply $C_1 A_1 B_0 X_0 = (\hat{C}_0 \hat{B}_0 - C_1 B_1) X_1$, i.e. a strong relationship between the values of X_0 and X_1.

Pros of being sequential. [79] proved that S6 – with no need for positional embeddings – can simulate any autonomous nonlinear dynamical system evolving in the direction $i \rightarrow i + 1$. This result is rooted in more general statements regarding Turing Completeness of RNNs [80, 81]. Indeed, in language modeling, Mamba is used *without positional embeddings* [54], in contrast to Softmax Attention without masking which requires positional embedding information capture distance information within X[4].

Cons of being sequential. While in the language modeling is convenient to drop positional embeddings, in the 2D and 3D applications, the notion of "position" cannot be easily captured by 1D ordering in a sequence: when processing data X where each X_i relates to a precise position in space, the output Y crucially depends on the chosen order the X_is are arranged into – in contrast with softmax attention (see propositions above). In a point cloud, we might order along the principal axis in 3D space and feed point clusters one at a time along these axes [60] or along an octree-determined path [62]. The output of S6, in this case, still depends on the processing order, regardless of the inclusion of additional positional embeddings in X and despite the potential bidirectional application of such models.

In this work, my goal is to work towards a principled strategy for processing point clouds with inherently sequential models such as Mamba. I first describe in-depth one existing approach [60] in Sec. 5.4.3 and then propose a patch reordering strategy that is able to match or improve performance compared to existing approaches, without requiring positional embedding but *relying solely on the sequential patch ordering* I introduce. This both re-

[4][64] recently proved that causal self-attention can instead recover positional information in 1D structures. Yet, modern practice still adopts positional embeddings by default also in this setting.

veals sensitivity to Mamba in sequence construction and potential for future developments using our strategy.

PointMamba Strategy

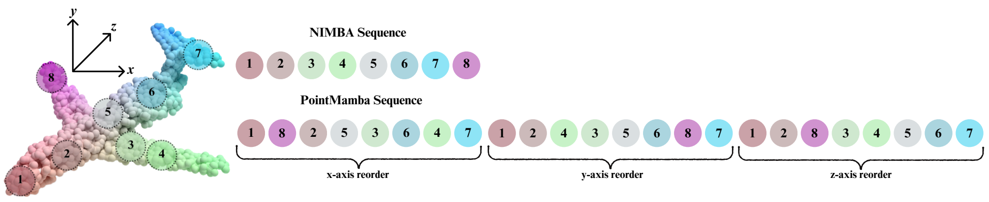

Figure 5.1: Ordering Strategy of NIMBA and PointMamba. The image is taken from the original work of NIMBA [20].

Despite the conceptual difficulty in processing 3D data with a sequential models, several approaches have been tested in the literature (see Sec. 5.2). Here I present the strategy proposed by PointMamba [60]: Following the notation of Sec. 5.4.1, centers $\{C_i\}_{i=1}^{n_c}$ are first sorted along each axis (x, y, z) independently, resulting in three separate *ordered* sequences: $(C_i^x)_{i=1}^{n_c}$, $(C_i^y)_{i=1}^{n_c}$, and $(C_i^z)_{i=1}^{n_c}$. For each axis-sorted sequence, the corresponding patch embeddings $\{\mathbf{p}_i\}$ and center embeddings $\{\mathbf{c}_i\}$ are obtained and then concatenated in the three orders above to form the input sequence X.

This strategy allows for successful processing, as I report in Sec. 5.5, yet has several weaknesses:

- The method results in the sequence length being tripled, introducing redundancy and negatively affecting efficiency.
- Centers that are close in the 3D space may not be adjacent in the sequence, which can affect the model's robustness and ability to capture spatial relationships effectively
- As I show in Sec. 5.5, this method is highly sensitive to the presence of positional embedding information. While this is common in standard attention-based architectures, it is less natural in Mamba-based models. In addition, it increases number of parameters and introduces additional redundancy.

NIMBA Strategy

To address the limitations of the PointMamba strategy, I introduce the **NIMBA** approach, which is designed to more effectively preserve geometric relationships by ensuring that consecutive centers in the 1D sequence input to the model remain close in 3D space. The NIMBA strategy is built around the principle of local proximity preservation:

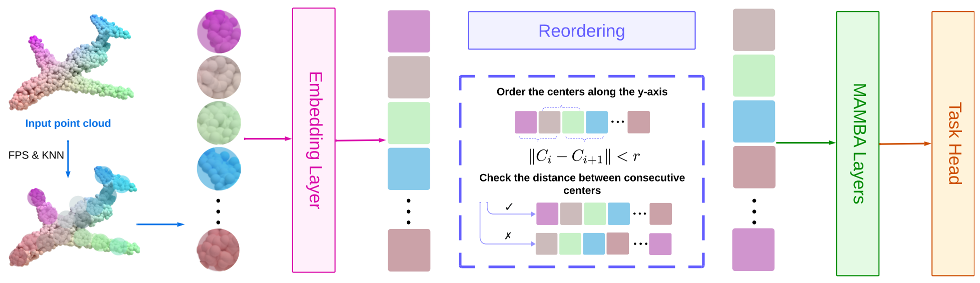

Figure 5.2: Overview of NIMBA pipeline. The image is taken from the original work of NIMBA [20].

1. *Initial Axis-Wise Ordering:* Initially, centers are flattened by ordering them along the y-axis. While any initial order could be used, I chose the y-axis because empirical results showed that this ordering helps reduce the computational cost in the subsequent phase.

2. *Proximity Check:* I scan the previously obtained sequence and iteratively check the distance between the current and the next center. If the distance exceeds a predefined threshold r, I search for a center along the sequence that is sufficiently close to the starting center and place it next to it. If no suitable center is found within the threshold, the sequence moves on to the next center without modification. This process ensures that consecutive centers in the sequence are within a distance of r.

Mathematically, the proximity check can be expressed as $\|C_i - C_{i+1}\| < r$, where $\|\cdot\|$ denotes the Euclidean distance in 3D space. The choice of r is critical: a large threshold value (e.g., $r \geq 2\sqrt{3}$, the diagonal of a unit cube) ensures that the sequence remains close to the initial order, as the condition will always be satisfied. On the other hand, a small threshold (e.g., $r = 0$) is computationally expensive, as each center would need to be compared with all others in the sequence, leading to an ordering identical to the initial axis-wise order, since no centers would be considered close enough to trigger any reordering. In my experiments, I found $r = 0.8$ to strike a good balance between the quality of the reordered sequence and computational cost. Other than efficiency reasons, the choice of the threshold r is related to the nature of point cloud datasets. Indeed, in ModelNet and Scanobject datasets the objects are contained in a $[-1,1]^3$ cube. The literature confirms that this comes from the nomalization step, which is a standard procedure in similar works and datasets. The choice of r can be interpreted as a portion of the distance between the center of the scene and the border of the scene, which should be 40%. Figure 5.2 illustrates the complete NIMBA pipeline.

Figure 5.1 shows the sequence created from the two strategies. When comparing to the PointMamba approach, NIMBA does not rely on positional embeddings and does not replicate the sequence: the reordering strategy proposed leverages the spatial relationships of points, allowing the model to rely only on the patch embedding, thus enhancing accuracy and stability.

5.5 Experimental Results

In this section, I give more details on the training setting that I used for my experiments.

To make a fair comparison, I followed the work proposed by PointMamba [60], PointMAE [34] and PCT [32] and I show the specific settings for the 3 different datasets and tasks: object classification on the synthetic ModelNet40 in Table 5.2, object classification on ScanObjectNN in Table 5.3 and segmentation on ShapeNet in Table 5.4. Instead of fine-tuning a pre-trained model, I trained from scratch to better highlight the differences between each setting. The main hyperparameter that I tuned in all the experiments reproduced is the learning rate and I did it with the criteria already explained in Chapter 4. I used a model of dimension 384 with 12 encoder layers and 6 heads across all experiments. There are some slight differences in the number of points sampled, number of patches and number of points per patch across the experiments, but still following the works mentioned above. For all methods I reproduce, I applied the same tuning efforts as for NIMBA, repeating all experiments three times and reporting the results as mean accuracy $\pm$ standard deviation.

Configuration	**Details**	**Value**
Model Configuration	Transformer Dimension	384
	Num. of Encoder Layers	12
	Num. of heads	6
Points Configuration	Num. of Points	1024
	Num. of Patches	64
	Num. of Point per Patches	32
Training settings	Optimizer	AdamW
	Learning Rate	1e-4
	Weight Decay	5e-2
	Scheduler Type	Cosine
	Num. of Epochs	300
	Num. of Warm-up Epochs	10
	Batch Size	32
	Seeds	0, 123, 777

Table 5.2: Training configuration for classification on ModelNet40. The table is taken from the original work of NIMBA [20].

Configuration	Details	Value
Model Configuration	Transformer Dimension	384
	Num. of Encoder Layers	12
	Num. of heads	6
Points Configuration	Num. of Points	2048
	Num. of Patches	128
	Num. of Point per Patches	32
Training settings	Optimizer	AdamW
	Learning Rate	5e-4
	Weight Decay	5e-2
	Scheduler Type	Cosine
	Num. of Epochs	300
	Num. of Warm-up Epochs	10
	Batch Size	32
	Seeds	0, 123, 777

Table 5.3: Training configuration for classification on ScanObjectNN. The table is taken from the original work of NIMBA [20].

Configuration	Details	Value
Model Configuration	Transformer Dimension	384
	Num. of Encoder Layers	12
Points Configuration	Num. of Points	2048
	Num. of Patches	128
	Num. of Point per Patches	32
Training settings	Learning Rate	1e-4
	Weight Decay	5e-2
	Scheduler Type	Cosine
	Num. of Epochs	300
	Num. of Warm-up Epochs	10
	Batch Size	16
	Seeds	42, 123, 777

Table 5.4: Training configuration for classification on ShapeNetPart. The table is taken from the original work of NIMBA [20].

5.5.1 Object Classification

I evaluate our proposed model, NIMBA, against various baseline models on multiple object classification benchmarks, including ModelNet [73] and three versions of ScanObjectNN (OBJ-BG, OBJ-ONLY, and PB-T50-RS from [74]). Results are summarized in Table 5.5.

Model	Backbone	Param. (M)↓	Accuracy (%)↑			
			ModelNet	OBJ-BG	OBJ-ONLY	PB-T50-RS
PointNet [82]*	Neural Network	3.5	89.2	73.3	79.2	68.8
PointNet++ [83]*	Neural Network	1.5	90.7	82.3	84.3	77.9
PCT [32]*	Transformer	2.9	90.17	-	-	-
Point Mamba†	Mamba	12.3	92.08 ± 0.16	87.80 ± 0.72	87.20 ± 0.88	82.20 ± 0.45
(My)	Mamba	12.3	**92.10 ± 0.14**	**89.06 ± 0.42**	**89.29 ± 0.23**	**83.91 ± 0.38**
Point-MAE†	Transformer	22.1	92.30 ± 1.02	86.77 ± 0.91	86.83 ± 0.78	81.23 ± 0.77
PointMamba†	Mamba	23.86	92.08 ± 0.19	88.01 ± 0.77	86.49 ± 0.49	83.01 ± 0.82
(My)	Mamba	23.86	**92.10 ± 0.14**	**89.80 ± 0.36**	**89.76 ± 0.37**	**84.21 ± 0.65**

Table 5.5: Accuracy on classification tasks. Different scales are reported. * are values reported from the PointMamba paper [60], while † are my reproducing choosing the best-performing learning rate for each model and task. The table is taken from the original work of NIMBA [20].

Transformer-based Models. Transformer-based models such as PCT and Point-MAE achieve competitive accuracies on these benchmarks. However, NIMBA surpasses these models by up to ≈ 2% on several datasets while using fewer parameters. Importantly, NIMBA achieves these improvements without employing positional embeddings.

Mamba-based Models. For Mamba-based architectures, the baseline PointMamba achieves strong performance. NIMBA exceeds PointMamba across all datasets, with accuracy improvements of up to 1.5%. Additionally, when scaling up to 23.86 M parameters, NIMBA continues to enhance its performance, surpassing the larger PointMamba model. These results demonstrate that NIMBA effectively leverages additional parameters to improve accuracy while maintaining efficiency.

Models	Param.(M)↓	Time (m)↓	
		ModelNet	ScanObjectNN
PointMamba†	17.4	500	240
NIMBA(My)	**17.4**	**430**	**200**

Table 5.6: Training time comparison after 300 epochs. The table is taken from the original work of NIMBA [20].

Training Efficiency. Beyond accuracy, I also assess the training efficiency of NIMBA compared to PointMamba. As shown in Table 5.6, NIMBA reduces the training time by ≈ 14% on ModelNet and ≈ 17% on ScanObjectNN after 300 epochs of training. This improvement in training speed further highlights the efficiency of our model.

Overall, NIMBA outperforms both transformer-based and Mamba-based baseline models without relying on positional embeddings, demonstrating its effectiveness in object classification tasks.

5.5.2 Part Segmentation

Models	Param.(M)↓	Cls. mIoU(%)↑	Inst. mIoU(%)↑
PointNet*	-	80.39	83.7
PointNet++*	-	81.85	85.1
Point-MAE†	27.1	83.91 ± 0.43	85.7 ± 0.23
PointMamba†	17.4	83.37 ± 0.17	85.07 ± 0.12
NIMBA(My)	**17.4**	**84.36 ± 0.06**	**85.54 ± 0.05**

Table 5.7: Performance comparison on the ShapeNetPart segmentation task. * indicates values reported in the PointMamba paper [60], while † denotes my reproduced results using the best-performing learning rates for each method. The table is taken from the original work of NIMBA [20].

I evaluate NIMBA on the part segmentation task using the ShapeNetPart dataset. As shown in Table 5.7, I report the mean IoU (mIoU) for both class-level (Cls.) and instance-level (Inst.) metrics. NIMBA achieves higher Cls. mIoU compared to both Transformer-based and Mamba-based models and demonstrates competitive performance in Inst. mIoU. Specifically, NIMBA outperforms the Mamba-based baseline, PointMamba, by $\approx 1\%$ in class-level accuracy while maintaining similar instance-level performance. Since all models are tuned to best independently, I attribute this performance boost to our improved reordering strategy.

5.5.3 Ablations

Here, I present a series of ablation studies to investigate the impact of the different components even further. In particular, in Sec. 5.5.3 I show and compare the effects of positional embedding, in Sec. 5.5.3 I test the robustness of models when noise is applied and in Sec. 5.5.3 I show how a bidirectional implementation of Mamba affects performances. All the ablations were made on the classification task on the ScanObjectNN dataset OBJ-BG variation.

Effect of Positional Embedding

To investigate the impact of positional embedding, I conducted a series of experiments comparing transformer, Mamba, and hybrid models. As shown in Table 5.8, performance generally declines when positional embedding is removed, affecting both models with attention blocks and Mamba-based models. This includes PoinTramba, which, despite outperforming NIMBA under normal conditions, relies heavily on positional embedding. Without it, NIMBA achieves better results. I observed that many Mamba-like models using positional embedding often replicate sequences or add bidirectionality to maintain performance. My hypothesis is that this phenomenon is due to

Models	**Acc. with PE(%)↑**	**Acc. without PE(%)↑**	**Gap(%)↓**
Point-MAE[†]	86.77 ± 0.91	80.24 ± 0.87	6.53 ± 1.78
PointMamba[†]	87.80 ± 0.72	83.69 ± 0.76	4.11 ± 1.48
PoinTramba[†]	**92.42 ± 0.48**	86.46 ± 0.34	5.96 ± 0.82
NIMBA(My)	89.80 ± 0.36	**88.12 ± 0.54**	**1.68 ± 0.90**

Table 5.8: Influence of Positional embedding (PE) on performance. * indicates values reported in the PointMamba paper [60], while † denotes my reproduced results using the best-performing learning rates. The table is taken from the original work of NIMBA [20].

redundancy: when sequences are insufficiently meaningful, the model scans them multiple times for better information retrieval. In contrast, NIMBA's reordering strategy preserves sequence length and performs well without positional embedding.

Robustness

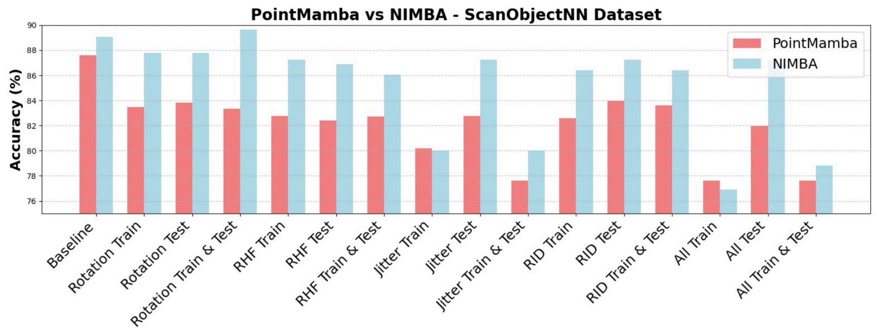

Figure 5.3: Results of applying noise to the training set, test set, or both. NIMBA demonstrates greater robustness compared to PointMamba, particularly when the noise does not alter the spatial distances between points, such as in the case of rotation. The image is taken from the original work of NIMBA [20].

To further explore the differences between PointMamba and NIMBA, I tested both models by applying the following noise injections to the input point clouds:

- Rotation: A random 3D rotation of the object;
- Random Horizontal Flip (RHF): A random flip along the horizontal axis;
- Jittering: Points in the point cloud are perturbed with Gaussian white noise;
- Random Input Dropout (RID): Points are randomly removed with a probability p;
- All: A combination of all the noise types listed above.

Each type of noise was applied to the training set, the test set, or both. As shown in Figure 5.3, my method NIMBA generally exhibits greater robustness to noise, particularly in the case of rotation, where there is even an improvement in performance. This confirms that the reordering strategy employed by NIMBA is resilient to noise that preserves pairwise distances between points, such as after a rotation.

Hydra

Models	**Param.(M)↓**	**Accuracy(%)↑**
PointMamba with hydra	12.85	86.23
NIMBA with hydra	**12.85**	**86.4**

Table 5.9: Results when substituting the Mamba block with the Hydra block in the architecture. The table is taken from the original work of NIMBA [20].

Building on previous works that utilize scanning different directions [61, 62, 71], I explored the impact of replacing the Mamba block with Hydra [84], a bidirectional extension of Mamba using a quasiseparable matrix mixer, in both PointMamba and NIMBA. Hydra scans sequences in both directions simultaneously, meaning PointMamba still processes a sequence of length $3N$, while NIMBA still processes length N. As shown in Table 5.9, performance generally declined in both cases, likely due to the shift to Hydra, which is based on Mamba2 [85]. I encourage future research to focus on optimizing Mamba2 in these contexts, as optimization remains a key challenge with such models.

5.6 Limitations and Future Work

While NIMBA successfully tackles several challenges in point cloud analysis with SSMs, some limitations persist. From an optimization perspective, the model shows limited improvement when scaled. Furthermore, when replacing the Mamba block with Mamba2 or integrating NIMBA into hybrid architectures, I observed a decline in performance, suggesting potential issues with integration. I encourage further research into optimizing SSMs and exploring their integration with transformer architectures for point cloud analysis. I believe this work provides a fresh perspective on applying non-transformer models in fields beyond natural language processing, emphasizing the potential of SSMs in 3D vision applications.

Chapter 6

Conclusion

In this work, I compare the capabilities of new deep recurrent models.

In the first part of this work, I investigate the reasoning capabilities of attention and recurrent-gated convolutional models in language modeling using the Associative Recall task as a proxy for language. However, since there is a gap in the performance of such models when trained on proper language, I studied in more depth the Multi-Query Associative Recall task, a more challenging version of AR that better approximate language modeling. At first sight, it seems that only attention models are capable of solving such a task, but this work highlights two key aspects to consider. Firstly, how crucial optimizing these models is to efficiently compress information in their hidden state and secondly, how attention and recurrent models benefit differently from scaling in depth and width. In fact, even if attention is capable of leveraging induction heads with a 2-layer architecture, it is not capable of solving the task with just 1-layer. However, the loss curve shows how the model still somehow tries to build induction heads during training. On the other hand, recurrent models benefit the most from scaling in width, by increasing their hidden state dimension that can compress information more easily.

In the second part of this work, I introduced NIMBA, a robust and principled approach for point cloud processing using state space models (SSMs). When using such casual models, a key challenge is to effectively convert a 3D set of data into a 1D sequence for proper analysis. To address this, I propose a spatially-aware reordering strategy that preserves spatial relationships between points. Differently from others, this method eliminates the need for positional embeddings and sequence replication, enhancing both efficiency and performance. My experimental results demonstrate that NIMBA matches or surpasses transformer-based and other Mamba-based models on benchmark datasets such as ModelNet, ScanObject, and ShapeNetPart in classification and segmentation tasks.

Chapter 7

Acknowledgments

Isaac Newton once said, “If I have seen further, it is by standing on the shoulders of giants.” This phrase deeply resonates with me since every achievement I have reached is built on the foundation of those who came before me and those who have supported and guided me along the way. This thesis, and indeed the person I have become, is a collective result of their contributions.

I thank the Politecnico di Torino for funding part of this journey and my advisor Prof. Lia Morra for her advice, constructive feedback, and guidance throughout these months.

I owe a special thanks to Dr. Antonio Orvieto, who first introduced me to the world of scientific research and inspired me to embrace it as a future path. His boundless curiosity and depth of knowledge were infectious, sparking my own enthusiasm. I am especially thankful for his trust and the freedom he gave me to explore ideas, fostering an environment of creativity and intellectual growth. My gratitude extends to the entire ELLIS and Max Planck communities, for the invaluable human and computational resources that made this journey and work possible. Particularly, thanks to the Deep Models and Optimization group, for their warm welcome, guidance, and unwavering support.

My appreciation goes further to Prof. Thomas Hofmann from ETH Zurich, my advisor abroad, for his confidence in my abilities, his enthusiasm for my research and his illuminating insights. Thanks to the Data Analytics lab for providing a vibrant, inclusive environment, filled with engaging discussions and invaluable suggestions that enriched this work immeasurably.

Outside of academic circles, I extend my gratitude to my ”less technical colleagues", my friends and roommates, who made my time abroad unforgettable. To Tübingen, thank you for immersing me in the richness of German culture and leaving me with fond memories. Special thanks to Culmannstrasse, both the permanent residents and exchange students, for sharing their cultures, laughs, and patience, creating lifelong memories. I truly hope our paths will cross again in the future.

Back home, my deepest thanks go to my Italian friends, who have been my constant pillars of support. To the San Remigio and Safa communities, thank you for helping me develop my character and interpersonal skills, traits I consider as vital as academic knowledge. To my high school and university friends, who stood by me even during the challenges of the COVID-19 pandemic. In particular, thanks to "Varie ed Alcool" for all the laughter, unforgettable nights, and support, which have kept my spirits high even while I was kilometres away.

I must also thank the incredible educators who inspired me from the very beginning. Thank you, Maestra Silvana, for sparking my curiosity and shaping me as a student. To Prof. Martinotti and Prof. Genna, your dedication to nurturing my passion for mathematics equipped me with the tools and confidence to explore this fascinating field.

I express my deepest gratitude to my families, yes, in plural. To my biological mother Pat and siblings, thank you for all the immense sacrifices you made and the unwavering love you gave me. To my foster parents, Daniela and Enrico, thank you for opening your hearts and home to me and for ensuring I never lacked anything. To my foster siblings, thank you for accepting me as a brother and for inspiring me to step outside my comfort zone and embark on this incredible journey abroad.

Lastly, To Georgiana, the love of my life, thank you for teaching me the meaning of love, for the unavailable day-to-day support even during these difficult months, and for the countless beautiful memories we've created together.

Bibliography

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[2] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2023. Available at https://arxiv.org/abs/2312.00752.

[3] Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2022.

[4] Daniel Y. Fu, Tri Dao, Khaled K. Saab, Armin W. Thomas, Atri Rudra, and Christopher Ré. Hungry hungry hippos: Towards language modeling with state space models, 2023. Available at https://arxiv.org/abs/2212.14052.

[5] Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

[6] Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. In *International Conference on Machine Learning*, pages 28043–28078. PMLR, 2023.

[7] Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. *arXiv preprint arXiv:2312.04927*, 2023.

[8] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014. Available at https://arxiv.org/abs/1410.5401.

[9] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. Available at https://arxiv.org/abs/1607.06450.

[10] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads, 2022. Available at https://arxiv.org/abs/2209.11895.

[11] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. Available at https://arxiv.org/abs/2104.09864.

[12] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. In *International Conference on Learning Representations*, 2020.

[13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks, 2013. Available at https://arxiv.org/abs/1211.5063.

[14] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33:1474–1487, 2020.

[15] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pages 5156–5165. PMLR, 2020.

[16] Albert Gu, Ankit Gupta, Karan Goel, and Christopher Ré. On the parameterization and initialization of diagonal state space models. *arXiv preprint arXiv:2206.11893*, 2022.

[17] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022.

[18] Chaofan Tao, Qian Liu, Longxu Dou, Niklas Muennighoff, Zhongwei Wan, Ping Luo, Min Lin, and Ngai Wong. Scaling laws with vocabulary: Larger models deserve larger vocabularies, 2024. Available at https://arxiv.org/abs/2407.13623.

[19] Sho Takase, Ryokan Ri, Shun Kiyono, and Takuya Kato. Large vocabulary size improves large language models, 2024. Available at https://arxiv.org/abs/2406.16508.

[20] Nursena Köprücü, Destiny Okpekpe, and Antonio Orvieto. Nimba: Towards robust and principled processing of point clouds with ssms, 2024. Available at https://arxiv.org/abs/2411.00151.

[21] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[22] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[23] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Neil Houlsby, Sylvain Gelly, Xiaohua Zhang, and Jakob Uszkoreit. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[24] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

[25] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[26] Gedas Bertasius, Heng Wang, and Lorenzo Torresani. Is space-time attention all you need for video understanding? In *ICML*, volume 2, page 4, 2021. Available at https://arxiv.org/abs/2102.05095.

[27] Zhan Tong, Yibing Song, Jue Wang, and Limin Wang. Videomae: Masked autoencoders are data-efficient learners for self-supervised video pre-training. *Advances in neural information processing systems*, 35:10078–10093, 2022.

[28] Ze Liu, Jia Ning, Yue Cao, Yixuan Wei, Zheng Zhang, Stephen Lin, and Han Hu. Video swin transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3202–3211, 2022.

[29] Limin Wang, Bingkun Huang, Zhiyu Zhao, Zhan Tong, Yinan He, Yi Wang, Yali Wang, and Yu Qiao. Videomae v2: Scaling video masked autoencoders with dual masking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14549–14560, 2023.

[30] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[31] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024.

[32] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R. Martin, and Shi-Min Hu. Pct: Point cloud transformer. *Computational Visual Media*, 7(2):187–199, April 2021.

[33] Xumin Yu, Lulu Tang, Yongming Rao, Tiejun Huang, Jie Zhou, and Jiwen Lu. Point-bert: Pre-training 3d point cloud transformers with masked point modeling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 19313–19322, 2022.

[34] Yatian Pang, Wenxiao Wang, Francis EH Tay, Wei Liu, Yonghong Tian, and Li Yuan. Masked autoencoders for point cloud self-supervised learning. In *European conference on computer vision*, pages 604–621. Springer, 2022.

[35] Xiaoyang Wu, Li Jiang, Peng-Shuai Wang, Zhijian Liu, Xihui Liu, Yu Qiao, Wanli Ouyang, Tong He, and Hengshuang Zhao. Point transformer v3: Simpler faster stronger. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4840–4851, 2024.

[36] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*, pages 9621–9630, 2019.

[37] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. *Advances in neural information processing systems*, 31, 2018.

[38] Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

[39] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[40] James Lee-Thorp, Joshua Ainslie, Ilya Eckstein, and Santiago Ontanon. Fnet: Mixing tokens with fourier transforms. *arXiv preprint arXiv:2105.03824*, 2021.

[41] Yifan Chen, Qi Zeng, Heng Ji, and Yun Yang. Skyformer: Remodel self-attention with gaussian kernel and nyström method. *Advances in Neural Information Processing Systems*, 34:2122–2135, 2021.

[42] Mitchell Wortsman, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Replacing softmax with relu in vision transformers. *arXiv preprint arXiv:2309.08586*, 2023.

[43] Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff. *arXiv preprint arXiv:2402.18668*, 2024.

[44] Jason Ramapuram, Federico Danieli, Eeshan Dhekane, Floris Weers, Dan Busbridge, Pierre Ablin, Tatiana Likhomanenko, Jagrit Digani, Zijin Gu, Amitis Shidani, et al. Theory, analysis, and best practices for sigmoid self-attention. *arXiv preprint arXiv:2409.04431*, 2024.

[45] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[46] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *arXiv preprint arXiv:2407.08608*, 2024.

[47] Soham De, Samuel L. Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, Guillaume Desjardins, Arnaud Doucet, David Budden, Yee Whye Teh, Razvan Pascanu, Nando De

Freitas, and Caglar Gulcehre. Griffin: Mixing gated linear recurrences with local attention for efficient language models, 2024. Available at https://arxiv.org/abs/2402.19427.

[48] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. *arXiv preprint arXiv:2312.06635*, 2023.

[49] Zhen Qin, Songlin Yang, Weixuan Sun, Xuyang Shen, Dong Li, Weigao Sun, and Yiran Zhong. Hgrn2: Gated linear rnns with state expansion. *arXiv preprint arXiv:2404.07904*, 2024.

[50] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *arXiv preprint arXiv:2406.06484*, 2024.

[51] Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xlstm: Extended long short-term memory. *arXiv preprint arXiv:2405.04517*, 2024.

[52] Karan Goel, Albert Gu, Chris Donahue, and Christopher Ré. It's raw! audio generation with state-space models. *International Conference on Machine Learning*, 2022.

[53] Eric Nguyen, Michael Poli, Marjan Faizi, Armin Thomas, Michael Wornow, Callum Birch-Sykes, Stefano Massaroli, Aman Patel, Clayton Rabideau, Yoshua Bengio, et al. Hyenadna: Long-range genomic sequence modeling at single nucleotide resolution. *Advances in neural information processing systems*, 36, 2024.

[54] Roger Waleffe, Wonmin Byeon, Duncan Riach, Brandon Norick, Vijay Korthikanti, Tri Dao, Albert Gu, Ali Hatamizadeh, Sudhakar Singh, Deepak Narayanan, et al. An empirical study of mamba-based language models. *arXiv preprint arXiv:2406.07887*, 2024.

[55] Yue Liu, Yunjie Tian, Yuzhong Zhao, Hongtian Yu, Lingxi Xie, Yaowei Wang, Qixiang Ye, and Yunfan Liu. Vmamba: Visual state space model, 2024. Available at https://arxiv.org/abs/2401.10166.

[56] Lianghui Zhu, Bencheng Liao, Qian Zhang, Xinlong Wang, Wenyu Liu, and Xinggang Wang. Vision mamba: Efficient visual representation learning with bidirectional state space model. *arXiv preprint arXiv:2401.09417*, 2024.

[57] Kunchang Li, Xinhao Li, Yi Wang, Yinan He, Yali Wang, Limin Wang, and Yu Qiao. Videomamba: State space model for efficient video understanding. *arXiv preprint arXiv:2403.06977*, 2024.

[58] Zhaohu Xing, Tian Ye, Yijun Yang, Guang Liu, and Lei Zhu. Segmamba: Long-range sequential modeling mamba for 3d medical image segmentation. *arXiv preprint arXiv:2401.13560*, 2024.

[59] Zeyu Zhang, Akide Liu, Ian Reid, Richard Hartley, Bohan Zhuang, and Hao Tang. Motion mamba: Efficient and long sequence motion generation with hierarchical and bidirectional selective ssm. *arXiv preprint arXiv:2403.07487*, 2024.

[60] Dingkang Liang, Xin Zhou, Wei Xu, Xingkui Zhu, Zhikang Zou, Xiaoqing Ye, Xiao Tan, and Xiang Bai. Pointmamba: A simple state space model for point cloud analysis, 2024. Available at https://arxiv.org/abs/2402.10739.

[61] Tao Zhang, Xiangtai Li, Haobo Yuan, Shunping Ji, and Shuicheng Yan. Point could mamba: Point cloud learning via state space model. *arXiv preprint arXiv:2403.00762*, 2024.

[62] Jiuming Liu, Ruiji Yu, Yian Wang, Yu Zheng, Tianchen Deng, Weicai Ye, and Hesheng Wang. Point mamba: A novel point cloud backbone based on state space model with octree-based ordering strategy. *arXiv preprint arXiv:2403.06467*, 2024.

[63] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5828–5839, 2017.

[64] Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers. *Advances in Neural Information Processing Systems*, 36, 2024.

[65] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

[66] Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

[67] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. Available at https://arxiv.org/abs/1810.04805.

[68] Peng-Shuai Wang. Octformer: Octree-based transformers for 3d point clouds. *ACM Transactions on Graphics*, 42(4):1–11, July 2023.

[69] Guangyan Chen, Meiling Wang, Yi Yang, Kai Yu, Li Yuan, and Yufeng Yue. Pointgpt: Auto-regressively generative pre-training from point clouds, 2023. Available at https://arxiv.org/abs/2305.11487.

[70] Xu Han, Yuan Tang, Zhaoxuan Wang, and Xianzhi Li. Mamba3d: Enhancing local features for 3d point cloud analysis via state space model, 2024. Available at https://arxiv.org/abs/2404.14966.

[71] Zicheng Wang, Zhenghao Chen, Yiming Wu, Zhen Zhao, Luping Zhou, and Dong Xu. Pointramba: A hybrid transformer-mamba framework for point cloud analysis, 2024. Available at https://arxiv.org/abs/2405.15463.

[72] Qingyuan Zhou, Weidong Yang, Ben Fei, Jingyi Xu, Rui Zhang, Keyi Liu, Yeqi Luo, and Ying He. 3dmambaipf: A state space model for iterative point cloud filtering via differentiable rendering, 2024. Available at https://arxiv.org/abs/2404.05522.

[73] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes, 2015. Available at https://arxiv.org/abs/1406.5670.

[74] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Duc Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data, 2019. Available at https://arxiv.org/abs/1908.04616.

[75] Li Yi, Vladimir G Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics (ToG)*, 35(6):1–12, 2016.

[76] Ameen Ali, Itamar Zimerman, and Lior Wolf. The hidden attention of mamba models. *arXiv preprint arXiv:2403.01590*, 2024.

[77] Jerome Sieber, Carmen Amo Alonso, Alexandre Didier, Melanie N Zeilinger, and Antonio Orvieto. Understanding the differences in foundation models: Attention, state space models, and recurrent neural networks. *arXiv preprint arXiv:2405.15731*, 2024.

[78] Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. Combining recurrent, convolutional, and continuous-time models with linear state space layers. *Advances in neural information processing systems*, 34:572–585, 2021.

[79] Nicola Muca Cirone, Antonio Orvieto, Benjamin Walker, Cristopher Salvi, and Terry Lyons. Theoretical foundations of deep selective state-space models. *arXiv preprint arXiv:2402.19047*, 2024.

[80] Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 440–449, 1992.

[81] Stephen Chung and Hava Siegelmann. Turing completeness of bounded-precision recurrent neural networks. *Advances in neural information processing systems*, 34:28431–28441, 2021.

[82] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation, 2017. Available at https://arxiv.org/abs/1612.00593.

[83] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space, 2017. Available at https://arxiv.org/abs/1706.02413.

[84] Sukjun Hwang, Aakash Lahoti, Tri Dao, and Albert Gu. Hydra: Bidirectional state space models through generalized matrix mixers, 2024. Available at https://arxiv.org/abs/2407.09941.

[85] Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality, 2024. Available at https://arxiv.org/abs/2405.21060.