On Memory and Generalization in the Era of Linear Recurrence

Anonymous authors
Paper under double-blind review

Abstract

Memory is crucial for the ability to store and retrieve prior knowledge when that information is gathered as a continuous stream that cannot be processed all at once. For decades, various types of artificial recurrent neural networks (RNNs) have been designed and improved to handle sequential data, incorporating memory in different ways. Transformers have become the most widely adopted architecture to deal with sequential data, while more recently structured state-space models (SSMs) and linear RNNs were put forward for their improved computational efficiency. While these families of models have been studied on various synthetic and real-world tasks, the generalization abilities of these newer models remain a topic of ongoing exploration. In particular, there is a gap in the current literature regarding the length generalization of models on sequence modeling tasks, both across models and across tasks. For models, while numerous studies have investigated the generalization of RNNs and Transformers to longer sequences, there is not much work devoted to such studies for SSMs or linear RNNs. Regarding tasks, one limitation of current works is their focus on formal language tasks for studying the generalization of sequence modeling. In contrast, the deep learning literature often introduces a variety of other tasks to assess the specific capabilities of deep learning models on sequential data. In this paper, we take a step toward addressing this gap by comparing the generalization abilities of all three families of algorithms across tasks that impose different memory requirements and are of special interest to the deep learning community, namely, copying tasks, state tracking tasks, and counting tasks. Our results show that despite their great efficiency, state space models seem to be less able than the non-linear recurrent models to generalize to longer sequences.

1 Introduction

The ability to effectively learn from sequences forms a significant part of deep learning research, given the wide amount of data in the real world that comes under this form. Various advancements have been made throughout the years by first determining limitations in existing methods, which led to developments in architecture and algorithms to overcome them. From the early recurrent networks, such as vanilla recurrent neural networks (RNNs) (Jordan, 1986; Rumelhart et al., 1986) which suffered from the *vanishing/exploding gradient* problem (Bengio et al., 1994; Pascanu et al., 2013), came variants that better overcame these through various modifications, such as adding gating components (Hochreiter & Schmidhuber, 1997; Cho et al., 2014; Chung et al., 2015), designing better initialization of recurrent weights (Le et al., 2015; Tallec & Ollivier, 2018; van der Westhuizen & Lasenby, 2018), parametrizing weights as orthogonal matrices (Henaff et al., 2016; Jing et al., 2019) and using non-saturating activation functions (Chandar et al., 2019). Then, the introduction of Transformers (Vaswani et al., 2017) addressed the issue of parallelizability and enabled the learning of dependencies of any length, thanks to the attention mechanism (Bahdanau et al., 2015) and led to a large shift towards this new architecture. However, due to the quadratic complexity of the attention mechanism, with respect to the sequence length, the sequence length that Transformers can process is practically limited. Finally, state space models (SSMs) (Gu et al., 2021; 2022b) were developed with the claim of solving these remaining issues through the specific initialization of parameters of the recurrence matrix (Gu et al., 2020;

2022a), leading to an increased surge in interest in linear recurrence (Orvieto et al., 2023; Qin et al., 2023; De et al., 2024; Beck et al., 2024) under the belief that such a paradigm can avoid prior limitations.

Understanding how these different models learn from sequences remains an open area of discussion. However, an implicit requirement often assumed is the need to memorize information from the input for later use. This has an analogous parallel within human reasoning, where information seen along a temporal axis quickly fades away and is possibly lost, if not immediately relevant. To overcome this, humans utilize *memory* (Miller, 1956), using internal modules for short or long-term storage (Atkinson & Shiffrin, 1968; Baddeley & Hitch, 1974) of information that can then be extracted once a signal is received that the stored information is required. However, in deep neural networks, it remains a subject of research to explore the particular role of memory. In an attempt to understand the use of memory in sequence modeling, (Deletang et al., 2023) consider a range of recurrent neural networks with various ways of incorporating memory, as well as Transformers, and compare their performance on a set of formal language tasks that differ in their memory requirements in the context of the theory of automata and formal languages. As a method of evaluating whether the trained RNN has learned the correct algorithm to solve the corresponding task, they analyze the extrapolation capability of the trained models by looking at their generalization to longer sequences than the longest they had seen in the training set. Interestingly, they find remarkable similarities between the behavior of the RNN models and their corresponding automata.

While in their work, as well as in related research on the generalization of RNNs, such as (Wang & Niepert, 2019), the tasks considered are those typically examined in the context of automata theory and formal languages, in this paper, we are concerned with the tasks that are of particular interest and relevance to neural networks studies, namely, copying memory, state tracking and counting. Some of these tasks have emerged from the interest in modeling complex sequential data with deep neural networks, with all being crucial for benchmarking and evaluating different deep learning algorithms. Our interest in these tasks stems from their representation of significant challenges in sequential models. The copying memory task (Hochreiter & Schmidhuber, 1997; Arjovsky et al., 2016) is a benchmark extensively used in the literature to assess how sequential models address the vanishing gradient problem and handle long sequence modeling. Moreover, this task requires the memorization of a sequence of data, making it an ideal test for neural networks with robust memory structures. While many variants of RNNs have successfully solved the copying task, few studies test whether they learn the correct algorithm and hence, generalize. On the other hand, certain state tracking tasks may be too complex, in terms of circuit complexity, for linear sequential models like structured state space models (S4) to solve (Merrill et al., 2024). Therefore, various versions of these tasks, with varying levels of difficulty, are used to benchmark linear sequential models, such as S4-types, against non-linear ones. Regarding generalization performance, while some earlier works, such as (Deletang et al., 2023), have studied the capability of Transformers and sequential neural networks on state tracking tasks, their sequential models are limited to non-linear RNNs; only a few recent studies have investigated the ability of certain linear sequential models to generalize to longer sequences. Sarrof et al. (2024) study the expressiveness of SSMs on specific formal languages, uncovering a design choice that renders the state transition matrices of these models non-negative and consequently hinders their generalization, even on some solvable state tracking tasks such as parity. Subsequently, in a concurrent work to ours, Grazzi et al. (2024) suggest modifications to some of these models, resulting in a more general transition matrix; while their proposed solution partially improves model generalization, some models including Mamba, still fail to generalize on certain hard state tracking tasks. Exploring the generalization capabilities of a wider range of the linear sequential models and tasks can provide valuable insights into the performance of these models compared to non-linear ones. Therefore, aside from the gap in task types, another aspect of the generalization studies in learning models that requires further exploration, is the variety of models that have been examined. Our work aims to address these gaps by employing a similar approach to the one used by Deletang et al. (2023). Finally, the counting task, in the context of timing (Gers & Schmidhuber, 2000; Gers et al., 2002), demonstrates the ability of sequential models to measure the temporal distance between events in sequences with rhythmic patterns. This capability is crucial for tasks such as music processing.

In the following sections, after reviewing some related works, we define a set of representative tasks, each necessitating different types of memory usage to be solved effectively. Next, we consider a set of commonly used neural network models with different ways that memory is utilized in them for sequence modeling, and

evaluate these architectures and tasks under various settings to understand how memory is employed in learning from sequences. In summary, our contributions are as follows:

- We define a memory taxonomy that applies to the deep learning synthetic tasks in our study, while relating those tasks and their memory requirements to corresponding tasks in formal language theory.
- We conduct a set of experiments to examine if and how the use of memory in various neural network architectures alters the learning process for different tasks, focusing on both the model's ability to learn and to generalize to more challenging settings.
- Following the claim of Merrill et al. (2024) that SSM models, such as S4 and Mamba, and Transformers share a similar inability to learn hard state tracking tasks due to their parallelizable nature, we empirically demonstrate that, even in generalization, these state space models exhibit behaviour similar to Transformers on Regular Language tasks (Deletang et al., 2023). Specifically, even for solvable state tracking tasks where SSM models have been successfully trained, they are unable to generalize to longer sequences.

Overall, our work contributes to this growing field by extending the studies of memory and generalization to include more recent sequential models within the current state of deep learning research.

2 Related Work

Memory in Neural Networks Memory plays a fundamental role in human cognition (Atkinson & Shiffrin, 1968) and can be classified into multiple hierarchies (Cowan, 2008), in particular *short-term* (or *working*) and *long-term* memory. Recurrent neural networks stood out due to their ability to learn and carry out complicated transformations of data over extended periods of time, as well as the potential to simulate arbitrary procedures with proper construction (Siegelmann & Sontag, 1995). They handle variable-length sequences by having a recurrent hidden state whose activation at each time is dependent on that of the previous time through a general update equation

$$\boldsymbol{h}_t = \boldsymbol{f}(\boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t) \tag{1}$$

where $\boldsymbol{h}_t$, $\boldsymbol{x}_t$ stand for the hidden state and the input vector at time t, $\boldsymbol{W}, \boldsymbol{U}$ are weight matrices parameterizing the RNN and $\boldsymbol{f}$ is an activation function of choice. However, due to the gradient scaling problems, the hidden state acts only as a short-term memory (Miller, 1956) and is limited in the duration and quantity of the stored information. By contrast, long-term memory can hold more information for extended time (Ratcliff, 1990). Attempts to alleviate this issue have primarily consisted along the direction of architectural design, such as the long short-term memory (Hochreiter & Schmidhuber, 1997) and gated recurrent units (Cho et al., 2014). Later, in an attempt to integrate better memory capabilities in sequential models, explicit memory modules (Graves et al., 2014; Gulcehre et al., 2017; 2018; Chandar et al., 2019) have been successfully explored to extend the capabilities of sequential models. This class of models, known as memory-augmented neural networks (MANNs), utilizes external memory, usually a matrix, to store information. A controller within the network learns to both read from and write to this external memory. This paradigm provides the model with greater flexibility to retrieve past information, forget, or store new information in the memory. Yet, these also pose challenges, in particular the additional complexity of managing an external memory structure and designing efficient algorithms for memory interactions, leaving it underexplored how memory explicitly helps with sequence modeling tasks.

Memory and Extrapolation to unseen examples In addition to helping RNNs overcome gradient scaling issues and achieve faster convergence on sequence modeling tasks (Graves et al., 2014; Chandar et al., 2019), memory cells have been shown in literature to significantly improve the generalization of memory-augmented models to unseen samples, such as samples with sequence lengths longer than those seen during training. Graves et al. (2014) first showed that their Neural Turing Machine (NTM) with an LSTM controller not only converges faster to a lower minimum than LSTM for the memory-intense task of copying memory, but also that in contrast to LSTM, the trained NTM can perform copying on sequences of more

than twice longer than the ones seen during training. Wang & Niepert (2019) then speculated that the poor extrapolation capabilities of RNNs stem from a lack of regularization in their hidden space, leading to the memorization of data rather than accurately storing the sequence state at each step. They propose a state regularization technique with a state extraction method for the automaton corresponding to the trained RNN and empirically show the benefit of this process for the generalization to out-of-distribution examples on tasks requiring some levels of memorization, such as balanced parenthesis (BP) task. One implicit assumption here is that the memory cells are required for the model to extrapolate, and state regularization optimizes the model's use of both its hidden state and its memory. Therefore, they consider the LSTM model for that task. The intuition comes from the fact that the balanced parenthesis task is classified as a context-free language in the Chomsky hierarchy, requiring a push-down automaton to express arbitrary depths of nested parentheses. Their experiment reveals that regularization prompts the LSTM to use its memory cell, rather than the hidden state, to track the nesting depth, akin to how the push-down automaton uses its stack. Consequently, a regularized RNN with a memorization mechanism becomes particularly advantageous for this task.

Finally, Deletang et al. (2023) conducted an exhaustive empirical study on the relationship between the architecture of sequence models and their ability to generalize to longer sequences. Their study spans a wide range of tasks and the corresponding state-of-the-art architectures, which they expect to better fit the task based on the Chomsky hierarchy, showing that while for higher levels of the hierarchy, different types of RNNs encounter difficulties in extrapolating beyond certain lengths, all of them, including the basic RNN and LSTM, generalize to a significant degree to longer lengths on regular language tasks such as parity and modular sum. This confirms that vanilla RNNs are capable of simulating finite state automata.

Counting with Neural Networks Counting is a trivial yet crucial skill possessed by the human mind, and can be split into three distinct groups: item counting in arrays, arithmetic, and event sequencing (Noël, 2009; Logie & Baddeley, 1987). The greatest utility of counting for real-world applications are tasks that require timing and sequence modeling, such as queuing, contextual awareness, and time series predictions. Learning to count requires the use of various strategies involving working memory, defined in cognitive literature as the temporary storage and processing of information (Logie & Baddeley, 1987). In the past, it was observed that recurrent models such as LSTMs that were extended by 'peephole connections', allowing them to observe their internal states, were able to time and count various target signals (Gers & Schmidhuber, 2000). More recently, it was shown that the popular Transformer models are also able to count, however, they are limited by the dimension of the model, and when performing counts on the most frequent element, single-layer Transformers are unable to learn this objective (Yehudai et al., 2024).

In the next section, we categorize the memory requirements for solving the deep learning tasks considered in this work and highlight their relationship to the categorization of related tasks in formal language theory.

3 Task-Based Memory Taxonomy

Significant research has been conducted on the memory requirements of tasks in deep learning architectures. One well-established approach to formalizing memory classification in machine learning is through the Chomsky hierarchy, which hierarchizes formal language tasks based on their complexities and the type of automata that can solve them. For instance, finite automata correspond to regular language tasks with limited memory requirements, whereas pushdown automata, with their stack-based memory, can handle more complex context-free languages. By drawing analogies between neural networks and different types of automata, we can comment on the networks' capabilities in performing specific tasks and, therefore, their expressivity (Weiss et al., 2018; Deletang et al., 2023). However, it is important to note that there is no one-to-one correspondence between neural networks and automata. A key reason is that automata have discrete states, while the hidden states of recurrent neural networks exist within a continuous space. This lack of exact mapping between the two classes of models means that empirical testing is necessary to validate the analogies and conclusions drawn about neural network capabilities[1]. In line with this argument, in this section, we review various levels of memory required for different learning tasks. We discuss the actual tasks

[1]Aside from the well-known works on extracting automata from trained RNNs, Wang & Niepert (2019) show how to extract a finite set of automata states being learned by RNNs during training.

considered in this study and their memory requirements in order to build a better taxonomy to distinguish our analysis. Notice that the categorization of memory in learning tasks presented here, including the titles of our categories, is based on our interpretation, drawing from both machine learning and neuroscience literature.

3.1 Memory-Free

The most basic tasks are *memory-free* tasks, where the output token only depends on the current input token and making the prediction does not require any memory of the past tokens (no history). Since each output only depends on the current input, any non-zero recurrences (introducing some form of memory) would introduce errors. As a result, memory-free tasks are not considered in this study.

3.2 Stateful Memory

We define the first type of memory requirement for sequences as *stateful* memory tasks. In this setting, it is possible to predict the next token based solely on the current *internal state* and the current input token. That is, if the internal state is a sufficient statistic for the history, then the problem is Markovian. Therefore, the primary learning problem in these scenarios becomes learning such statistics. Stateful memory is the type of memory required for problems where an internal state with a suitable constant size is enough to learn such statistics for a sequence regardless of its length.

Such tasks are closely related to regular languages in formal language theory, which need an automaton with the lowest level of memory to recognize them, i.e., a finite state automaton. [2] Stateful memory tasks include state tracking tasks such as bit parity tracking. In bit parity tracking, the model is given a sequence of 0s and 1s and must determine the current parity state of the sequence, either 0 or 1 (even or odd parity).

$$010101101 \xrightarrow{\text{Correct States}} 011001001$$

In this paper, experiments on stateful memory tasks are conducted on algebraic groups of different computational complexities, such as $\mathbb{Z}_{60}$, $\mathbb{A}_4 \times \mathbb{Z}_5$, and $\mathbb{A}_5$, which are studied by Merrill et al. (2024). Solving these tasks requires understanding the current state, which is either a position on a circle with k positions in the case of the $\mathbb{Z}_k$ groups or permutation of k numbers for $\mathbb{A}_k$ groups, and combining it with the current input token (which is a number of steps to move or an order to permute the system) in order to predict the correct output token (a new position or permutation). The first time step of the sequence tells to take an action from some initial state. In the second step, the model must learn to apply the next permutation to the previous output. Given a sequence, the target output is the correct position or permutation given the sequence of actions applied in order. Merrill & Sabharwal (2023) suggest that any architecture that can parallelize computation over a sequence inherently lacks the ability to represent languages of a specific complexity [3](problems that are NC^1-hard), including $\mathbb{A}_5$. As such, we are motivated to consider these tasks as examples of scenarios that only require stateful memory, since the output is always deterministically decided based on the previous state and the incoming input, and can still be difficult to solve due to circuit complexity considerations.

In these tasks with stateful memory requirements, sequences do not need to be long, as the relationships that are essential for learning the underlying model are short. Theoretically, training on all possible sequences of two tokens, consisting of an initial permutation, a permutation to execute on the internal state, and the final outcome permutation, is sufficient to learn the correct internal state to generalize to longer sequences. As the sequence length increases, the difficulty of the prediction also increases, since intermediate errors will compound and result in an incorrect final token prediction, if the internal state representation is incorrect.

State tracking tasks are representative of model capabilities in narrative understanding, such as discourse understanding or entity tracking (Kim & Schuster, 2023) and hence are significant in benchmarking different sequential models in the current state of language model research.

[2] One subtle difference is that in formal language theory, most tasks are defined in the context of acceptance and rejection, while the tasks of our interest are of the transduction type. The way we relate these two types of problems is similar to how Deletang et al. (2023) explain in their work.

[3] We use the term language as defined in Formal Language terminology.

3.3 Stable Memory

While stateful memory tasks do not require keeping a single element in memory for a very long time, this is the case of what we define as *stable* memory tasks. These tasks could depend on a single token, but that token has to be kept in memory for a potentially long period before the information is used as output. While they may not require learning a complex internal state representation (they could just copy the input token), they do require learning what must be stored in memory and keeping it in memory in order to learn the task. These tasks are distinct from stateful tasks in two key ways. Firstly, by definition, they necessitate a memory mechanism capable of retaining information for a long time. Secondly, the state size might need to expand depending on the amount of information that needs to be stored. This second characteristic implies that a constant-size state is inadequate for stable memory problems *with arbitrary sequence lengths*. Consequently, the formal language counterparts for these tasks reside higher in the Chomsky hierarchy than Regular Languages, including context-free and context-sensitive languages, which require additional memory structures such as stack and tape in their respective automata. However, note that in the specific cases where the sequence lengths are upper-bounded, the problem can be solved by a finite state automaton.

Examples of tasks demanding stable memory are the copying (Hochreiter & Schmidhuber, 1997; Arjovsky et al., 2016) and denoising (Jing et al., 2019) tasks. In the copying task, the model is given a sequence of S tokens from a predefined vocabulary, followed by a sequence of T noise tokens. After that, a specific indicator token tells the model to reproduce the initial S tokens from the beginning of the sequence. An example is shown below

$$\underbrace{\texttt{X}\ \ \texttt{Y}\ \ \texttt{Z}}_{\text{Sequence to copy of length } S}\qquad \underbrace{\texttt{[B]}\ \ \texttt{[B]}\ \ \texttt{[B]}}_{T \text{ Blank tokens}}\qquad \underbrace{\texttt{[C]}}_{\text{Indicator to start copying}}\qquad \underbrace{\texttt{X}\ \ \texttt{Y}\ \ \texttt{Z}}_{\text{Target Sequence}}$$

The denoising task, also called selective copying, is similar to copying task, except that the noise tokens are scattered in between the elements of the sequence of random tokens to be copied (still resulting in a sequence of $S + T$ tokens).

$$\underbrace{\texttt{X}\ \ \texttt{Y}\ \ \texttt{[B]}\ \ \texttt{[B]}\ \ \texttt{Z}\ \ \texttt{[B]}}_{S+T \text{ Interspersed Tokens}}\qquad \underbrace{\texttt{[C]}}_{\text{Copying indicator}}\qquad \underbrace{\texttt{X}\ \ \texttt{Y}\ \ \texttt{Z}}_{\text{Target Sequence}}$$

These tasks have been widely used to evaluate the capacity of models to learn long-range dependencies in sequences. Learning the denoising task is considered more challenging because the model must learn a more complex filtering/ignoring mechanism, as shown by Gu & Dao (2024). While those blanks may initially overfill the network memory in both tasks, they clearly indicate the end of the sequence to memorize in the copying task but not in the denoising task. In both cases, the memory of the non-blank tokens must be stable over the processing of the blank tokens. That is, the activation should not decay nor be altered by blank inputs. The gradient must also freely flow through time to learn the task correctly. If the memory is stable, the network should generalize to longer sequences of blanks. Intuitively, generalizing to memorize more tokens is expected to be more challenging, because from the perspective of Formal Language theory, it examines the ability of the model to handle the context-sensitive aspect of the task.

3.4 Counting Memory

Another critical role of memory is to perceive duration in order to learn to produce some output at a specific moment in time. This requires an understanding of counting and/or timing. We define this as *counting memory*. This type of memory resides between the stateful and stable ones defined earlier. While counting and timing tasks require noting time alongside learning the model's state, the required memory is simpler than the stable memory: instead of storing some specific content, the number of certain items or time steps need to be tracked and stored. The counterparts in formal languages are counter languages which aside from regular languages include some context-free and context-sensitive languages.

A simple yet representative task can be derived from classical and temporal conditioning (Gallistel & Gibbon, 2000) where, models are autoregressively trained with *on/off* (or 0/1) signals of varying lengths. The objective is to learn the duration of the *on* signal, which is fixed, to accurately predict when it will turn *off*. This duration, known as the interstimulus interval (ISI) in the conditioning literature, is interleaved with a random interval of *off* tokens, called the intertrial interval (ITI). The Networks that successfully learn to count

should focus on learning the ISI duration, as its end is fully predictable, whereas the beginning of the ISI is impossible to predict precisely. A crucial aspect of this timing task is that the actual task (timing the ISI) can start anywhere in the sequence, and the end to be predicted depends entirely on stable counting or time tracking from the ISI onset. The memory must then be fully reset to handle the next ISI in the sequence. In animals, the task does not get more difficult or easier as long as the ITI/ISI ratio remains constant (Gallistel & Gibbon, 2000). However, for most neural networks, a longer ITI should increase the training difficulties.

4 Experiments

In this section, we empirically evaluate the performance of different types of sequential models on the tasks described in Section 3. Given the nature of the tasks discussed, it follows that the varying inductive biases inherent in different neural network architectures can enable some to outperform others. In Appendix A, we describe the architectures we use in our experiments: RNN (Rumelhart et al., 1986), LSTM (Hochreiter & Schmidhuber, 1997), GRU (Cho et al., 2014), NRU (Chandar et al., 2019), Transformer (Vaswani et al., 2017), S4D (Gu et al., 2022a) and Mamba (Gu & Dao, 2024), and briefly discuss their memory properties, while additional implementation details are provided in Appendices B, D and F.1. We assess the models' abilities to both learn the tasks and generalize to out-of-distribution (OOD) examples, as a means to probe whether the models have learned the correct algorithm. In our state tracking, copying and denoising experiments, the OOD examples are sequences longer than the training samples.

4.1 State Tracking

State tracking experiments use the same setting as in (Merrill et al., 2024), specifically the use of groups $\mathbb{Z}_{60}$, $\mathbb{A}_4 \times \mathbb{Z}_5$ and $\mathbb{A}_5$. For all our experiments, we control the number of parameters in each layer to approximately 3 million parameters per layer. Explicit hyper-parameters are provided in Appendix D.

We train each model for up to 500 epochs, with early stopping if performance reaches a desired threshold (set at 90% accuracy) or fails to reach 1% accuracy after half the allocated training time. A model is deemed to learn the task if it achieves the threshold performance before the training budget elapses. Performance on a task (in this case, a group $g \in \{\mathbb{Z}_{60}, \mathbb{A}_4 \times \mathbb{Z}_5, \mathbb{A}_5\}$) is measured by the model's accuracy at predicting all elements across the entire sequence. The model is considered to have predicted a sample correctly if it can predict all tokens exactly within the sequence. Since all samples have the same length, overall performance on the task is measured by the number of correctly predicted samples within a held-out test set.

When training on sequences of length k, we first create a dataset of elements from the selected group of length k. We then randomly select a maximum of 10^6 elements from the dataset, which is further split into a training and testing split. Following Merrill et al. (2024), we always include all examples for $k = 2$ in the training set.

When evaluating for extrapolation to longer lengths, the number of parameters per layer is kept constant between models. But, we use the minimal number of layers necessary to achieve the threshold accuracy on the training length (empirically chosen as 90% in our experimentas). This number of layers can vary between different models for each task. Additionally, a model is considered to have 'extrapolated' to a length only if it simultaneously achieves the predefined threshold accuracy on that length as well as on all shorter test lengths.

Initial Long Sequence Results We first ask why learning group multiplication for the $\mathbb{A}_5$ group poses a challenge for models. Notably, while one-layer models can adequately model sequences of length $k = 2$, it is interesting that some models fail to learn the task when k increases-unless the model's depth (number of layers) is also increased accordingly. Given that the underlying group operations are deterministic in the token tagging structure, they should ideally be represented by a deterministic transition function that takes the current state and input token as inputs. Thus, our first question is whether there is a length limit to the effectiveness of one-layer models. To test this hypothesis, we significantly increase the value of k beyond 25, the limit tested by Merrill et al. (2024). For further comparison, we include a novel model they introduced, called the Input-Dependent S4 (IDS4). They claim IDS4 can solve the $\mathbb{A}_5$ class of problems due to the inclusion of input-dependent $\boldsymbol{A}$ transition matrices.

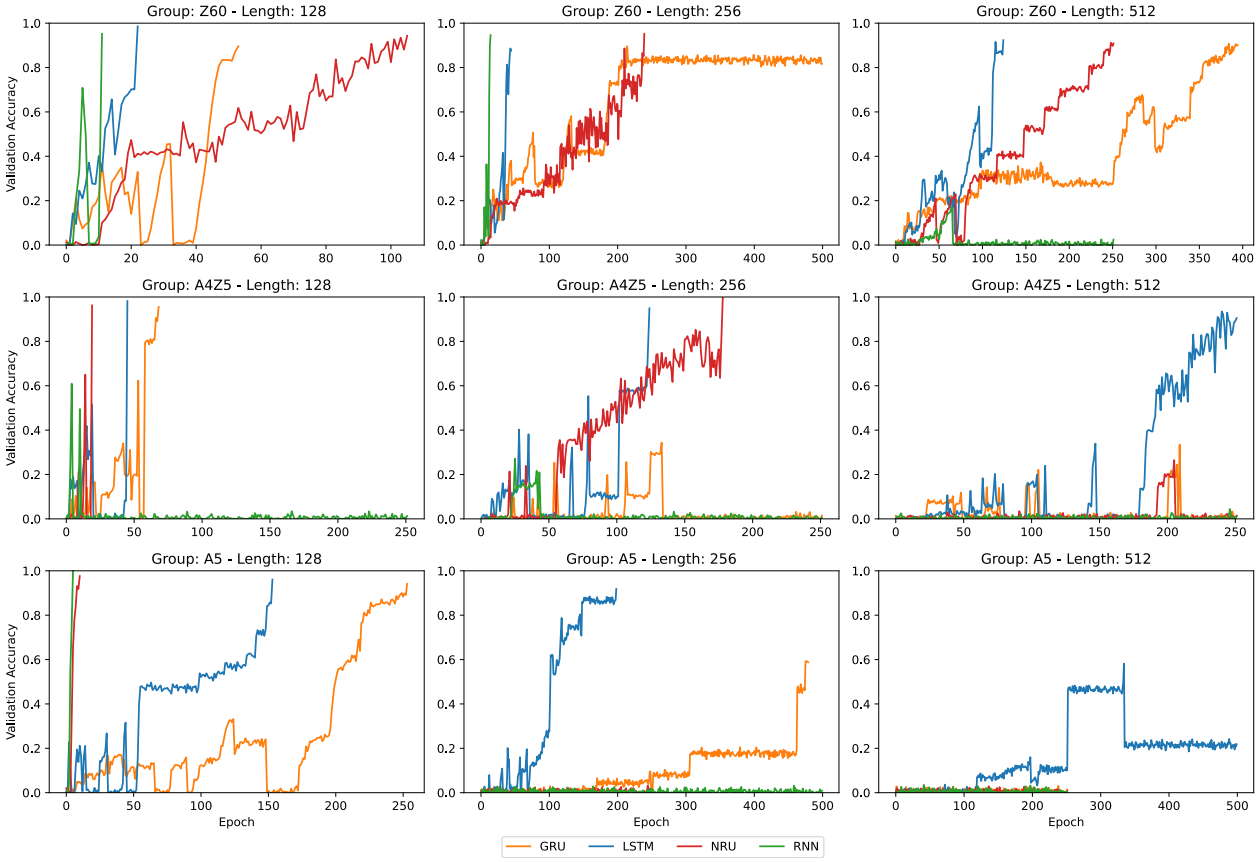

Figure 1: Ability of single-layer recurrent models to learn longer state tracking tasks. We observe that as the sequence gets increasingly long, even models that theoretically should be able to learn the task fail to do so, especially as the state tracking task becomes more complex (in terms of circuit complexity).

Task	$\mathbb{Z}_{60}$				$\mathbb{A}_4 \times \mathbb{Z}_5$				$\mathbb{A}_5$			
Model	RNN	LSTM	GRU	NRU	RNN	LSTM	GRU	NRU	RNN	LSTM	GRU	NRU
Length 128	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
256	✔	✔	✔	✔	✘	✔	✘	✔	✘	✔	✘	✘
512	✘	✔	✔	✔	✘	✔	✘	✘	✘	✘	✘	✘

Table 1: An extension of Merrill et al. (2024)'s on longer length state tracking tasks. (✔) indicates that a single-layer model attains $> 90\%$ accuracy on a held-out validation set (averaged over 5 seeds), (✘) means otherwise. We compare models suggested to have the ability to model the corresponding group structure of each task for arbitrary length. The IDS4 demonstrates significant numerical issues which renders it incapable of modeling sequences of longer than 20 tokens, hence we exclude it. Other models (Transformer, Mamba, S4D) cannot properly learn sequences with a single layer and are also excluded.

Table 1 shows that even recurrent models with non-linearities, which are claimed to have the ability to model arbitrarily long state tracking problems, have limits on all tasks being tested. Note that other models (Transformer, Mamba, S4D) cannot properly learn sequences with a single layer and therefore are excluded. IDS4, on the other hand, demonstrates significant numerical issues, which render it incapable of modeling sequences of longer than 20 tokens, and hence we exclude it as well. A comparison with other recurrent models further shows that those with explicit memory structures (LSTM and NRU) are better able to learn sequences of increasing length. Figure 1 further shows accuracy on a held-out validation set during training. Clearly, the direct learning trends of the different models change both between tasks as well as length. For

example, for sequences of length $k = 128$, the RNN manages to learn the task significantly faster than other models on the groups $\mathbb{A}_5$ and $\mathbb{Z}_{60}$, yet eventually the other models remain capable of learning on longer sequences, while the RNN cannot ($k = 256$ for $\mathbb{A}_5$ and $k = 512$ for $\mathbb{Z}_{60}$, respectively)[4]. This highlights the possibility that the inductive biases of such models, despite modeling the correct circuit complexity for these classes of problems, remain insufficient for such state tracking tasks.

	Task	$\mathbb{Z}_{60}$									$\mathbb{A}_4 \times \mathbb{Z}_5$									$\mathbb{A}_5$								
	Test Length	1X	2X	3X	4X	5X	64	128	256	512	1X	2X	3X	4X	5X	64	128	256	512	1X	2X	3X	4X	5X	64	128	256	512
Sequence Model	Mamba	✔	✘	✘	✘	✘	✘	✘	✘	✘	✔	✘	✘	✘	✘	✘	✘	✘	✘	✔	✘	✘	✘	✘	✘	✘	✘	✘
	Transformer	✔	✘	✘	✘	✘	✘	✘	✘	✘	✔	✘	✘	✘	✘	✘	✘	✘	✘	✔	✘	✘	✘	✘	✘	✘	✘	✘
	S4	✔	✘	✘	✘	✘	✘	✘	✘	✘	✔	✘	✘	✘	✘	✘	✘	✘	✘	✔	✘	✘	✘	✘	✘	✘	✘	✘
	IDS4	✔	✘	✘	✘	✘	✘	✘	✘	✘	✔	✘	✘	✘	✘	✘	✘	✘	✘	✔	✘	✘	✘	✘	✘	✘	✘	✘
	RNN	✔	✔	✔	✔	✔	✘	✘	✘	✘	✔	✔	✔	✔	✔	✘	✘	✘	✘	✔	✔	✔	✔	✘	✘	✘	✘	✘
	LSTM	✔	✔	✔	✔	✔	✔	✘	✘	✘	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✘	✘	✘	✘
	GRU	✔	✔	✔	✔	✔	✔	✘	✘	✘	✔	✔	✔	✔	✔	✔	✔	✔	✘	✔	✔	✔	✔	✘	✘	✘	✘	✘
	NRU	✔	✔	✔	✔	✔	✔	✘	✘	✘	✔	✔	✔	✔	✔	✔	✔	✘	✘	✔	✔	✔	✔	✘	✘	✘	✘	✘

Table 2: Evaluating the ability of models to extrapolate from their training length to longer sequences. Here, we provide an example where $k = 8$. The number of layers of each model depends on the ability to learn the training length (1X). (✔) indicates that the model can attain $> 90\%$ accuracy on a held-out validation set (averaged over 5 seeds), and (✘) means otherwise. We evaluate on extrapolation up to 5 times the training length and fixed lengths of 64, 128, 256, and 512.

Short-to-Long Extrapolation A particular feature of state tracking tasks is the deterministic nature of transitions. This means that if a model is learning the underlying transitions within the specific task, then learning properly from a short sequence can trivially extrapolate to sequences of any arbitrary length, as the rules of the transitions do not change within the group. This motivates us to explore the ability of different models to exhibit this feature. If a model fails to extrapolate to longer lengths, we hypothesize that it is learning something other than the underlying transitions. If this persists, then we can further contend that such models can suffer from difficulties when it comes to more complex state tracking tasks, especially those that have non-deterministic rules for their transitions.

Table 2 shows extrapolation performance from training on sequences of length 8. Interestingly, only non-linear recurrent networks are capable of extrapolating to any degree past the length of sequences used for training; this suggests that parallelizable models fail to properly learn the task. Meanwhile, we observe that non-linear recurrent models with explicit memory structures demonstrate improved extrapolation performance.

4.2 Copying

For this task, the input consists of a sequence of S random tokens, followed by T blank tokens, and finally an indicator token. The goal is to reproduce the S random tokens after seeing the indicator token. In practice, the input has a length of $T + 2S$, where we append another S blank tokens to the end of the input sequence. The model is trained to output blank tokens up to the $(S + T)$-th step (until it sees the indicator token). After seeing the indicator, it starts to output the random tokens (in order) when presented with S additional blank tokens.

As explained before, we are interested in two aspects of model performance: first, the training dynamics which we study through the evolution of accuracy over the random part of the sequence, i.e., the part of length S; secondly, the generalization of the model to unseen data; we do an extrapolation analysis and test the trained model on longer sequences than the ones seen during training. To compare different models, we consider two experimental setups. In the first one, we adjust all model hyper-parameters so that the number of parameters for all of them is similar. In the second setting, we consider models with the same hidden state size so that we can comment on how they may differently use the same hidden space. For both cases mentioned, we consider both 1-layer and 2-layer architectures.

[4]We conduct a grid search on a select set of hyper-parameters. Increasing model size (which we control for) may potentially change convergence rates of each model.

We have two different training setups which correspond to two different extrapolation experiments. In the first one, we train models on a mixture of data sequences with a fixed pattern length, but different lengths of the blank interval. This setup is later used to evaluate the capability of the trained model to extrapolate to sequences of the same pattern length, but longer blank interval length than in the training set. Training sequences have a fixed pattern length of 10 and contain up to 50 blank tokens, and we examine generalization to sequences containing up to 500 blank tokens.

In the second setup, we train models on a mixture of data sequences of a fixed blank interval length but varying pattern lengths. We then evaluate their extrapolation performance on sequences of the same blank interval length, but longer pattern lengths. Training sequences have blank tokens of lengths 50 and pattern lengths up to 10 and we examine the generalization of the models to sequences with up to 100 pattern lengths.

In all cases, for comparing the extrapolation capability of different architectures, as we observe the models achieve $> 99\%$ training ccuracy, we compare the generalization ability of the trained models in extrapolating to longer sequences to empirically investigate how different memory structures help with better extrapolation. However, some models did not learn the task within the maximum number of training steps set for our experiments, which is 150K. For those cases, we still illustrate their generalization performance on longer sequences, comparing it to their best-achieved accuracy at the end of the training.

We provide the list of hyper-parameters in Appendix B, including both global hyper-parameters, which remain fixed across all experiments, and those we set to different values to examine their potential effect on the results.

4.2.1 Mixed blank Interval Length

Figure 2 shows the results for training on mixed blank interval lengths but fixed pattern lengths for models with one and two layers, where different architectures have comparable sizes . As observed in Figure 2, while gated RNN models (GRU, LSTM, and NRU) successfully solve the task with both 1 and 2 layers, the vanilla RNN does not learn the task within our maximum number of training steps. S4D exhibits the fastest convergence, whereas the other SSM model, Mamba, solves the task only with 2 layers, within the given number of training steps. The Transformer's slow convergence is primarily due to the lack of Positional Encoding (NoPE). We did not include results for the Transformer with sine-cosine encoding because, although it learns the task very quickly, that type of positional encoding significantly reduces its extrapolation capability.[5] Note that all models were trained with the same learning rate of $1e-3$. From Figure 13 we observe similar results for the case where all models have the same hidden size. We still notice the slower convergence for the LSTM in this experiment compared with the larger LSTM in Figure 13.

Extrapolation to Longer Blank Interval Length Figure 3 shows the results of length generalization for different models with comparable sizes. The plots display the accuracy over the random numbers to be copied, which are of length 10 in all experiments. The dashed vertical blue line indicates the training range; sequences to the right of this line have not been seen during training and thus measure generalization. We observe that, despite its strong performance on the copying task, S4D fails to extrapolate to longer sequences. Conversely, gated RNNs, especially GRU and NRU, continue to perform well for longer distances. The 2-layer architectures exhibit similar behavior as the 1-layer ones. Finally, while Mamba takes longer than S4D to achieve perfect accuracy, it demonstrates superior performance in extrapolation. Overall, GRU with 2 layers shows the best extrapolation. In Figure 14, we observe similar results on the performance of architectures with the **same hidden size** on the same task.

4.2.2 Mixed Pattern Length

Figure 4 shows the result for training on mixed pattern length but fixed blank interval length on 1- and 2-layer models of comparable sizes. Figure 15 in the appendix shows the same experiment with models of the same hidden size.

[5]Also, the work by (Deletang et al., 2023) studies several other positional encodings including RoPE and ALiBi. We did not include them because Transformers were not the main focus of our work, and none of the encodings resulted in a good generalization performance on the duplicate string task, which is a formal language task related to our copying task.

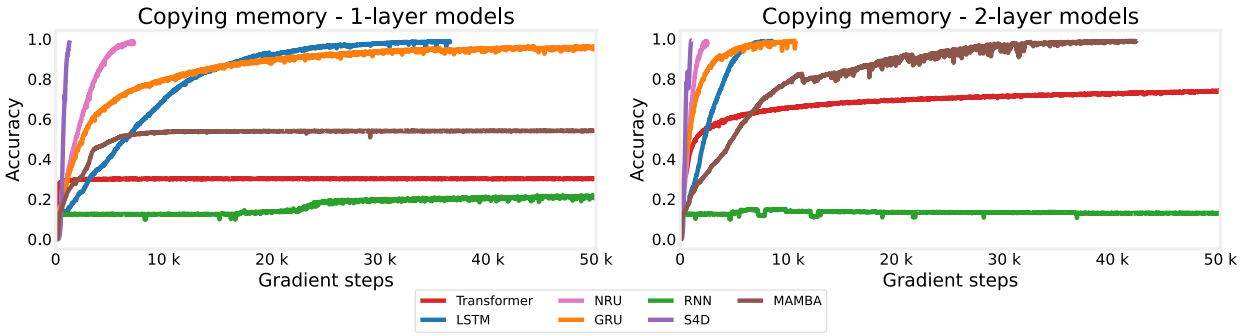

Figure 2: Learning dynamics of (left) 1- and (right) 2-layer models with **comparable number of parameters** for **copying memory** task trained on sequences with a **mixture of blank interval lengths**. While gated RNNs and S4D can solve the task with only one layer, 2-layer Mamba and Transformer take much longer to converge.

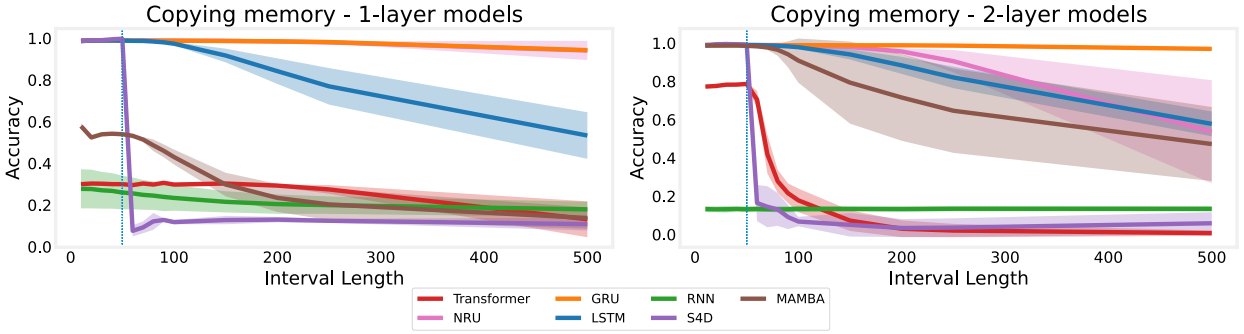

Figure 3: Extrapolation of (left) 1- and (right) 2-layer models with **comparable number of parameters** to longer **blank** interval length for **copying task**. The dashed vertical blue line is the training range. We especially observe that perfectly trained S4D fails to extrapolate to longer sequences. Also, GRU with 2 layers shows the best overall performance.

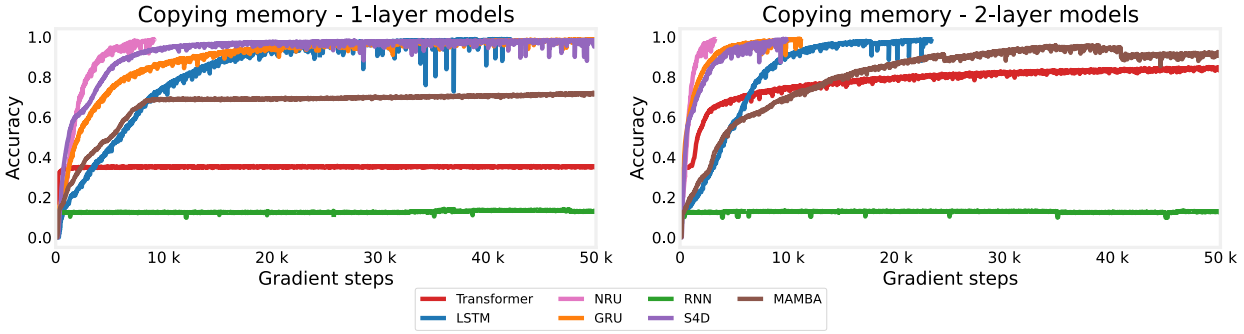

Figure 4: Learning dynamics for **copying memory** task with **similar number of parameters** trained on sequences with a **mixture of pattern lengths** up to 10. We observe that all gated RNNs, as well as S4D, are able to solve the task with only one layer, while Mamba and Transformer need a second layer.

Extrapolation to Longer Pattern Length Figure 5 shows the results of length generalization for different models with 1 and 2 layers where all models have the **same number of parameters**. Training sequences have noise tokens of lengths 50 and pattern lengths up to 10, and we examined the generalization of

the model to sequences with up to 100 pattern lengths. The main observation here is that all models that we have tried fail to extrapolate for pattern length twice the maximum training length and beyond. We especially observe that despite its fast convergence on the copying task, S4D is again the worst-performing model in extrapolating to unseen sequence lengths. For the 2-layer case where Mamba can solve the task, it interestingly shows better asymptotic extrapolation, on par with best performing RNN types of models. It should also be noted that our Mamba model is 4 times smaller than all other models due to some implementation restrictions that do not allow a hidden size larger than 256. Figure 16 in Appendix C shows the results for the same setting when all models have the **same hidden size**.

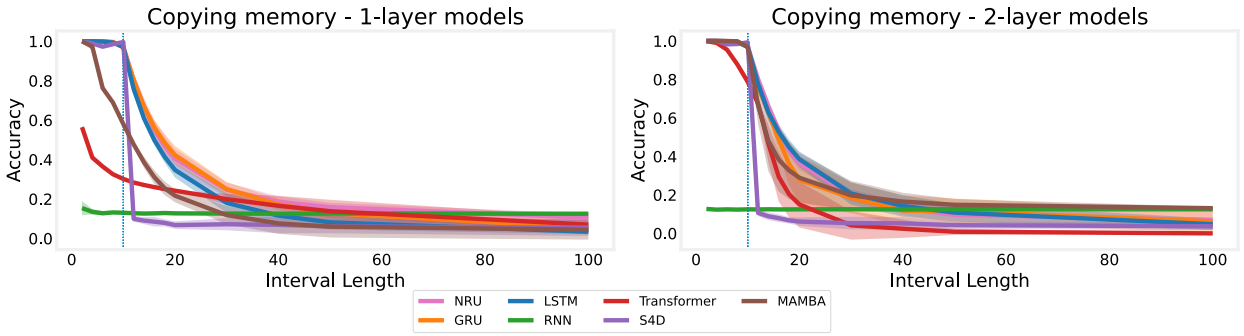

Figure 5: Extrapolation of 1- (left) and 2-layer (right) models with the **same number of parameters** to longer **pattern** length for **copying task**. The dashed vertical blue line is the training range. Notice that all models fail to extrapolate as we increase the pattern length to almost twice the maximum training length.

4.2.3 Summary of Results

The main result of this section is that, while some models can extrapolate to longer sequences with increased blank interval length, all of them struggle to generalize to sequences with longer pattern lengths. That is, they do not perform well when the amount of data they need to retain in memory increases. This happens while in principle the models have enough storage to store the information of those long sequences. This result aligns with the observations in (Deletang et al., 2023), where various sequential models, including Transformers, were tested on the related task of string duplication. This task falls within the context-sensitive class of the Chomsky hierarchy and requires an automaton with a memory tape. Their findings indicate that for this memory-intensive task, only Tape-RNN could extrapolate to longer strings, while other RNN models with less structured memory modules, such as LSTM and GRU, failed to generalize beyond their training sequence lengths.

4.3 Denoising

As explained in Section 3, this task is very similar to the copying memory task, with the only difference being the blank (noise) tokens scattered between the random tokens that should be copied rather than coming after them. Otherwise, the experiment setup and the model's evaluation during both training and generalization are exactly the same as described in the previous section for the copying task.

4.3.1 Mixed Noise Interval Length

Figure 6 shows the results for training on mixed blank interval lengths but fixed pattern lengths for models with one and two layers, where different architectures have the same number of parameters. Unlike the copying task, S4D demonstrates very slow convergence even with two layers, while the 2-layer Mamba successfully solves the task. Figure 17 presents the results for the same experiment using different models with the same hidden size. Interestingly, a vanilla RNN with 2 layers successfully solves the denoising task as well.

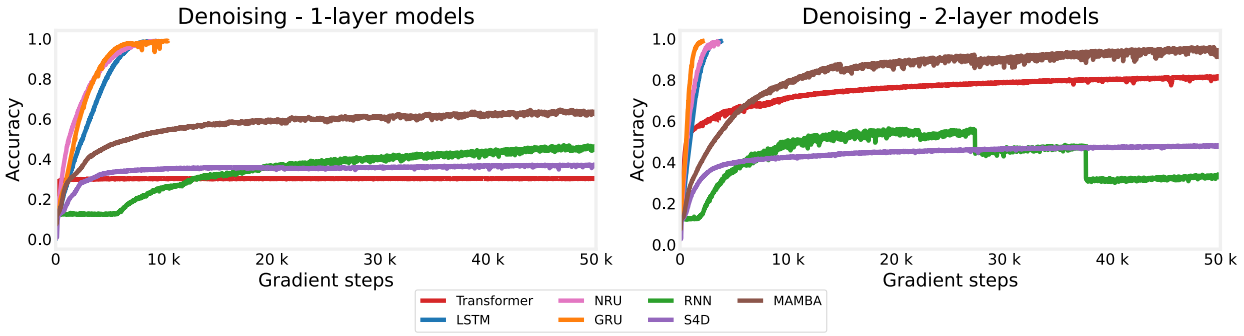

Figure 6: Learning dynamics for **denoising** task with **similar number of parameters** across models trained on sequences with a **mixture of blank interval lengths**. With a larger size, S4D still has a very slow performance even with 2 layers.

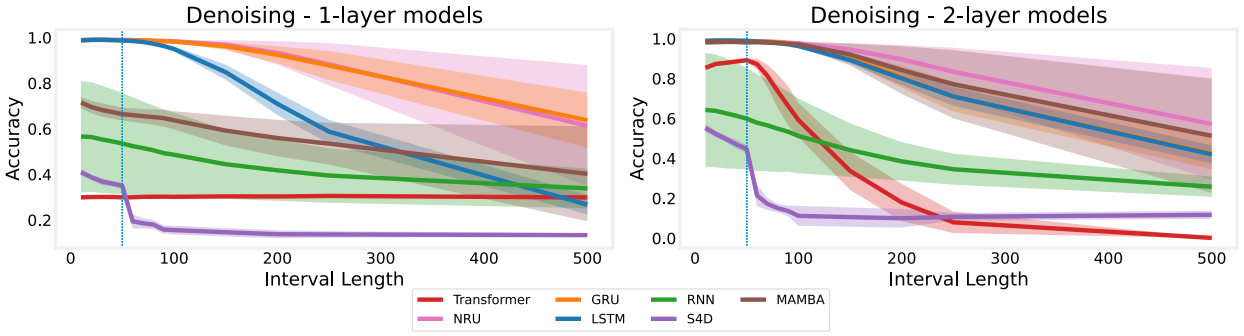

Figure 7: Extrapolation of (left) 1- and (right) 2-layer models with **comparable number of parameters** to longer **noise** interval length for **denoising task**.

Extrapolation to Longer Noise Length Figure 7 shows the extrapolation of the trained models to longer noise intervals. We especially observe that although 1-layer Mamba only partially solves the task, it is able to keep its performance for longer noise intervals. Also, while S4D cannot extrapolate to longer sequences, the 2-layer Mamba model behaves more like gated RNNs in terms of its extrapolation capability. In Figure 18, we present the results of a similar analysis for the case where all models have the same hidden size.

4.3.2 Mixed Pattern Length

Figure 8 shows the result for training on mixed pattern length but fixed blank interval length on 1- and 2-layer models of comparable sizes. On Figure 8, we again see that gated RNN models and 2-layer Mamba solve the task while S4D has very slow performance and Transformer (with NoPE) also converges much slower compared with RNN-type models. Comparing with Figure 19, under the same experimental setting with models of identical hidden size, we observe that the larger Mamba model solves the task at least twice as fast as the smaller model.

Extrapolation to Longer Pattern Lengths Figure 9 shows the result of length generalization for different models of **similar sizes** with one and two layers. Figure 20 illustrates the results of a similar analysis with models of the **same hidden size**. Notably, in both cases, we observe that increasing the number of layers does not help with better extrapolation.

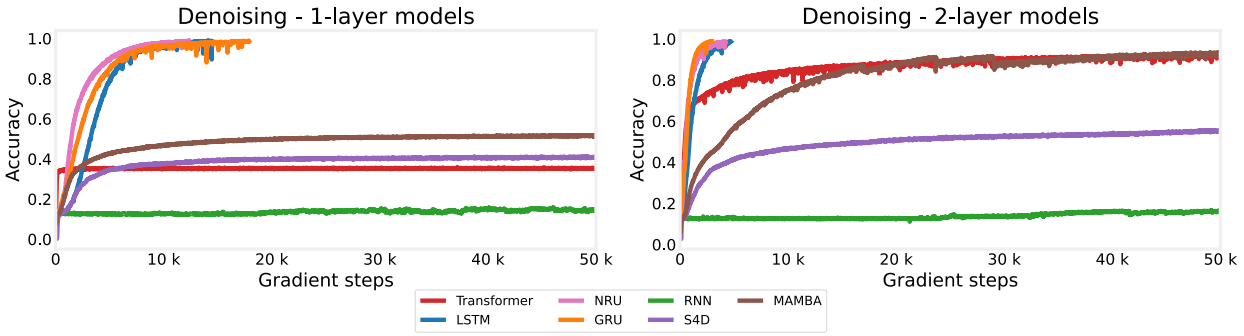

Figure 8: Learning dynamics for **denoising task** for models with **similar number of parameters** trained on sequences with a **mixture of pattern lengths**. While 2-layer Mamba solves the task, S4D shows a very slow performance. Again, RNN models beat all other architectures.

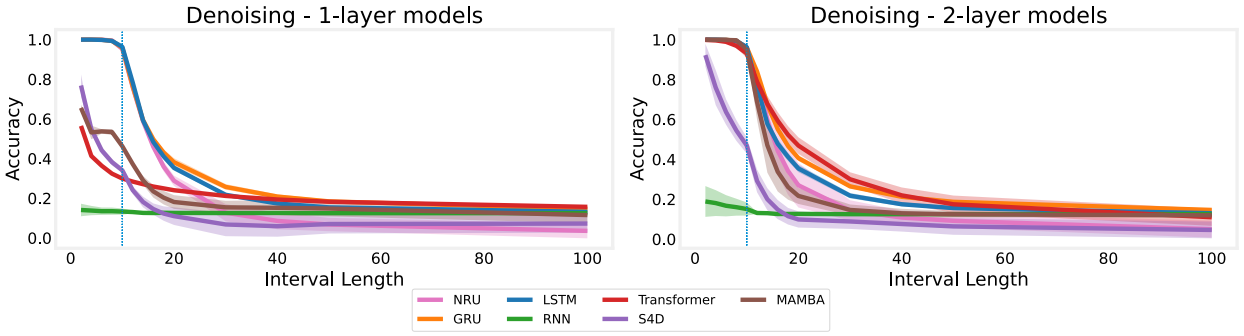

Figure 9: Extrapolation of (left) 1- and (right) 2-layer models with the **same number of parameters** to longer **pattern** length for **denoising task**.

4.3.3 Summary of Results

As in the copying task, the main result is that while several models achieve perfect performance for the sequence lengths seen during training, as we increase the pattern length at the test time, none of them can keep as good performance on any longer length, which could be expected from the task being memory intensive and hence requiring more specialized memory modules in the architecture. Regarding the capacity of the studied state space models in learning the task, the superior performance of Mamba compared to S4D could be attributed to its input-dependent weights, which enhance the filtering ability required for the denoising task. In contrast, S4D is time invariant, as explained in Section A.3.3.

4.4 Learning to Count

For the first task, we present models with a fixed-length stimulus (the ISI). The stimulus is removed for a random period (the ITI) and presented again, with this cycle continuing for a predetermined time. The ITI has a value of 0 and its duration is randomly picked from the set of the following intervals depending on the sequence length: $\{[20, 40], [60, 120], [100, 200]\}$. For the ISI, the signal is *on* with a value of 1 for one of the values of $\{10, 30, 50\}$ timesteps of the sequence, again depending on the sequence length. The length of the signals varies among 200 (short), 600 (medium), and 1000 (long) time steps. The signal is constructed by alternating ITIs and ISIs, where it always starts with an ITI followed by an ISI and this pattern repeats for the whole sequence length. Therefore, this memory task involves 2 components: the ability to reset at the onset of the stimulus and the ability to either count the duration of the ISI or to compute an offset time stamp to predict the ISI offset.

Counting requires models to realize the length of the ISI, as the ITI is random and therefore cannot be learned. The model's desired behavior requires counting to start once an onset is observed and stop at the desired offset. To assess whether the counting ability improves with the network scale, we further increase the hidden dimension from 8 to 64. However, our results suggest that the task is trivial for models with a hidden size of 8. Interestingly, even after training for much longer, up to a thousand epochs, the performance remained relatively the same, indicating that counting does not improve with increasing the capacity of the network.

Model		**RNN**		**LSTM**		**GRU**		**NRU**		**TF**		**S4D**		**Mamba**	
Hidden Size		8	64	8	64	8	64	8	64	8	64	8	64	8	64
Length	200	✔	✔	✔	✔	✔	✔	✔	✔	✘	✘	✔	✔	✘	✘
	600	✘	✘	✔	✔	✔	✔	✔	✘	✘	✘	✔	✔	✘	✘
	1000	✔	✘	✔	✔	✔	✘	✔	✘	✘	✘	✔	✔	✘	✘

Table 3: Model Performance on training length of 1000 epochs, hidden sizes 8 and 64, on all 200, 600, and 1000 timestep sequences. (✔) indicates that the model was able to learn the signal and (✘) indicates otherwise.

Predicting Signal Offset In Table 3, we report the ability to predict the desired offset for short, medium, and long ISIs. We consider a model successful if it predicts the ISI offset at least 3 out of 5 trials within the specific configuration, measured using the validation MSE.

First of all, regardless of the length of the ISI and the hidden size, both Mamba and Transformer fail to learn the task, whereas LSTM and S4D succeed in learning the signal across all configurations. Another observation is that, except for the RNN model on the medium ISI, all recurrent models and S4D perform well with a hidden size of 8. However, a larger hidden size of 64 not only fails to introduce any benefits, but also, in some cases, hinders performance, such as with NRU and GRU for long ISI.

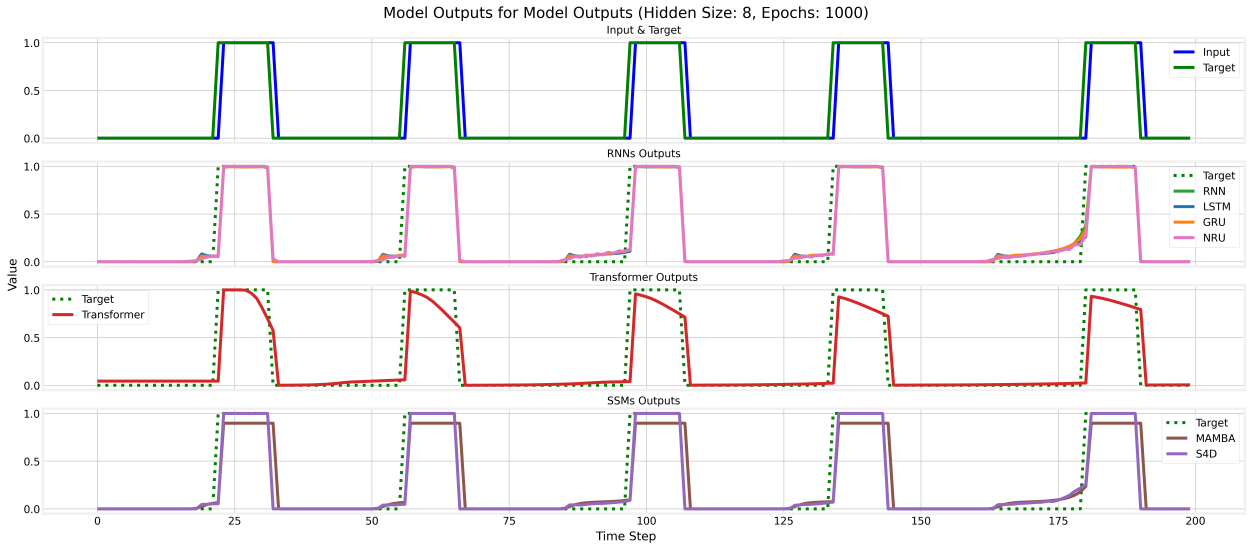

Figure 10: Model outputs for models with hidden size of 8, trained on 1000 epochs and a 200 sequence length. All recurrent models have slight anticipation for the onset and correctly predict the offset. The Transformer model has no anticipation for the onset, starts to offset early, and also misses the desired offset. S4D behaves similarly to the recurrent models, while Mamba, although it shows some anticipation of the onset, misses the desired offset timing.

Phase Alignment Tests A model's ability to generalize can be inferred by the ability to phase align, defined as resetting its memory after seeing a stimulus. Here, we conduct two phase alignment tests to

measure a model's learning ability. We run these tests on models trained on short signals, with a hidden size of 8 and 64, trained on 1000 epochs. We chose this configuration as most models can predict the desired offset in this setting. The results are similar for medium and long sequences.

Model		**RNN**		**LSTM**		**GRU**		**NRU**		**TF**		**S4D**		**Mamba**	
Hidden Size		8	64	8	64	8	64	8	64	8	64	8	64	8	64
Test	Two ISI	✘	✔	✘	✘	✘	✘	✔	✘	✘	✘	✔	✔	✘	✘
	Double ISI	✘	✘	✘	✘	✔	✘	✔	✘	✘	✘	✔	✔	✘	✘

Table 4: Phase alignment results for varying lengths on short signals sequences trained for 1000 epochs. (✔) indicates that the model was able to phase align at least 3 out of 5 times with the specific configuration, and (✘) indicates otherwise.

First, we evaluate whether models can reset their memory. To test this, a second ISI is placed after an initial ISI and a long ITI (longer than the ITI seen during training) at test time. This assesses the ability to consistently recognize the onset of a signal and begin counting. The desired behavior is thus for the model to correctly predict the end of the ITI and re-activate upon the appearance of the second ISI.

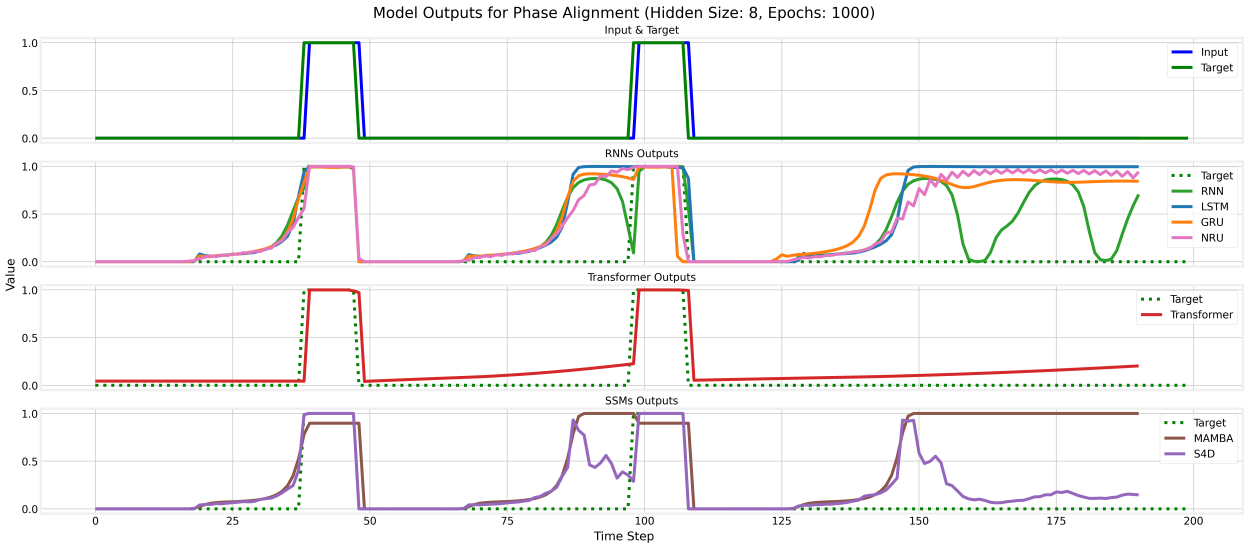

Figure 11: Phase alignment test results for two ISIs with an intermediate ITI. Recurrent models are unable to predict the offset of the second ISI. Transformer and Mamba miss the offset, while S4D can predict both offsets, indicating an understanding of the transition from ITI to ISI.

For hidden size 8, the NRU and S4D models succeed (Table 4). In contrast, for a hidden size of 64, only the RNN and S4D models complete the task. The RNN model was the only model that benefited from increasing the hidden size as it was able to predict the offset for the first and second ISI. The GRU and LSTM models similarly failed the test (Figure 11), with both models predicting the first offset but struggling with the anticipation of the onset and prediction of the offset of the second ISI. This suggests that they struggle to re-align their count to the ISI once introduced to longer ITIs. Increasing the hidden size hindered the NRU model from learning the tasks, namely, this increase introduced learning instabilities, where models produced NaN values. From Figure 11 we observe that the S4D model expects an ISI in between the longer ITI due to what it has seen during training, however, it turns off rather quickly and shows its ability to reset its memory. The Transformer and Mamba models fail in the same manner as the models with 8 hidden units.

The second test evaluates if models can correctly reactivate their counting mechanism when encountering consecutive ISIs. After completing the first ISI, the model should deactivate briefly, as learned during training,

then reactivate and continue counting for the second ISI's duration. The RNN, LSTM, Transformer, and Mamba models fail to do this (Figure 12). The GRU and NRU models succeeds with a hidden size of 8 but fail with a hidden size of 64. Only, the S4D model completes the test for both hidden sizes, retaining the ability to reset its memory when no stimulus is present.

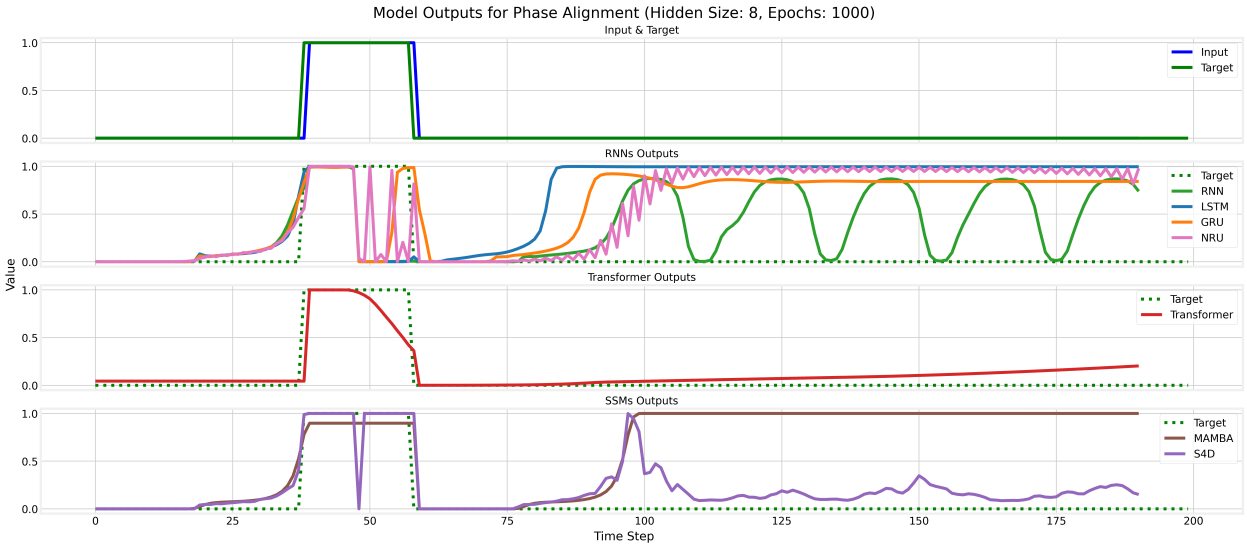

Figure 12: Phase alignment test results on two consecutive ISIs. We observe that only the NRU, GRU, and S4D models are able to reactivate after the first ISI, indicating that other models lack a complete understanding of the ISI.

Summary of Results From our experiments with the counting task, several key insights emerge. The Transformer and Mamba models consistently fail to learn the task, regardless of model size. The limitations with the Transformer are consistent with previous work where it was suggested that single-layer models, like ours, struggle with similar tasks (Yehudai et al., 2024). Mamba's poor performance, particularly in overestimating ISI lengths, contrasts with the S4D model, which excels across all configurations and signal lengths. It will be interesting to investigate if such gap in performances can be due to the positivity of Mamba's transition matrix. This property was recently shown to hinder the generalization of models such as Mamba in some state tracking tasks (Sarrof et al., 2024; Grazzi et al., 2024). Recurrent models such as GRU and LSTM perform well on most tasks, particularly in offset prediction. LSTM models enjoy success by leveraging their memory mechanism to handle ISI offsets as complexity increases, which aligns with prior findings in similar counting domains (Rivest et al., 2010; Suzgun et al., 2019). However, RNNs falter with longer ISI lengths, reflecting difficulties with long-term dependencies. Interestingly, NRU achieves success with a smaller hidden size but faces training instability as the hidden size increases. Across all recurrent models, increasing hidden size does not result in any notable improvement in generalization or task performance. Overall, S4D demonstrates superior generalization and performance, supporting prior claims of robustness in tasks involving long-term temporal dependencies, while other models exhibit varying degrees of success and limitations.

5 Discussion

In this section, we provide an additional overview of the motivation behind our research, our interest in the topic of neural network memory, and its relevance to our generalization studies. After highlighting the gaps in the current literature that our work addresses we review the main results of our experiments and the insights that our findings offer for future research.

5.1 Relating Memory and Extrapolation

Understanding deep learning models is crucial because these methods often operate as black boxes, making it difficult to interpret their solutions. It is essential to have insights and methods to interpret the model's solution to ensure that a model has learned the correct algorithm for a given problem. Extrapolation to out-of-distribution (OOD) samples serves as a significant indicator of whether the model has internalized the correct algorithm. On the other hand, in the framework of formal language theory in classical machine learning, there is a correspondence between different formal languages and various types of automata that differ in terms of their memory component. If an automaton with a finite number of states possesses the correct memory component for a task with given memory requirements, then in principle it should be able to learn the correct algorithm to solve the task and extrapolate to unseen examples. For example, pushdown automaton generalizes on the task of balanced parentheses to longer open parenthesis depth than seen during training, while a finite-state automaton cannot realize this generalization. While there is no direct equivalence between neural networks and automata, and many tasks of interest in deep learning research may not directly translate to formal language tasks, valuable insights can still be drawn. Similar arguments can be applied to analyze the suitability of deep learning models for different tasks. Therefore, categorizing the memory mechanisms of these models and studying their generalization capabilities across various problems is crucial. This approach provides essential insights into developing better models and designing more informative benchmarking tasks.

5.2 Limitations of Prior Studies

Research on the computational power of neural networks in relation to automata has been conducted for various recurrent neural networks (RNNs) across different formal language tasks (Weiss et al., 2018; Wang & Niepert, 2019; Deletang et al., 2023). However, with the recent development of alternative novel architectures, more complete studies are still largely absent from the current literature. In terms of OOD generalization, some of these new architectures have demonstrated exceptional extrapolation capabilities to longer sequences for specific tasks, such as associative recall and induction heads (Gu & Dao, 2024; De et al., 2024). On the other hand, various works have identified failure modes in other tasks, particularly different types of state-tracking (Sarrof et al., 2024; Grazzi et al., 2024). Since each of these synthetic tasks is representative of different capabilities of deep learning models, there is significant potential for further improvements in these models. Addressing these gaps and optimizing for various task requirements could lead to more versatile neural networks.

5.3 Our Results and Suggestions For Further Study

We present our main findings on the models' ability to learn and generalize in three specific tasks: group permutation state tracking, (selective) copying, and counting.

State Tracking Our observations here are three-fold. First, we observe that all models have limits both in terms of how they solve tasks as well as how they extrapolate to longer lengths. Especially, despite theoretical arguments in (Merrill et al., 2024) about single-layer SSMs being able to solve easy state tracking tasks, such as $\mathbb{Z}_{60}$ modular sum, in our experiments only non-linear models can solve those tasks with one layer for long sequences. Therefore, it is important to reconsider whether the failure of linear models on hard state tracking tasks is due to expressivity concerns or practical training limitations. This reevaluation is particularly important when considering arguments about the circuit complexity[6] of such problems, such as the widely accepted but unproven $\mathtt{TC}^0 \subset \mathtt{NC}^1$ assumption that underlies their justification.

Another significant observation from our experiments is that the solution of finite-layer SSM models on the easy state tracking tasks does not extrapolate to longer sequences, suggesting that the model has not learned the correct algorithm. Similar observations for Transformers were identified by Deletang et al. (2023), who noted that while these models could solve some regular formal language tasks, they failed to extrapolate. They hypothesized that this failure is due to the positional encoding of such architectures which makes the

[6]Circuit complexity relates to the width and depth of trees that represent boolean functions necessary to estimate a given problem accurately.

network activations out-of-distribution for longer sequences. However, other factors can also play a role. Liu et al. (2023) demonstrate how Transformers tend to learn shortcut solutions that, while valid, are less effective at generalizing to unseen data. This observation highlights a potential area of interest for SSMs.

Finally, this lack of generalization for linear sequential models holds also for the IDS4 model that Merrill et al. (2024) specifically constructed to solve even the hard state tracking tasks. While this may be unexpected from the perspective of (Merrill et al., 2024), it is consistent with a concurrent observation by Grazzi et al. (2024). Following Sarrof et al. (2024) who identified an important failure mode of current SSMs in the context of the solvable state tracking task of parity and attributed it to the positivity of the state transition matrix in non-time-invariant SSMs, such as Mamba, Grazzi et al. (2024) further investigated easy and hard group permutation tasks in a concurrent work. They showed that while modifying the state transition matrix of Mamba to include negative eigenvalues enables it to generalize on the parity task, the adjusted Mamba still poorly generalizes on the more challenging state tracking tasks involving group permutations. That the IDS4 is neither time-invariant [7] nor relies on a positive transition matrix, and still fails to extrapolate, adds to the evidence that resolving the eigenvalue issue as suggested by Sarrof et al. (2024) and Grazzi et al. (2024) alone may not fully address state-tracking challenges for SSMs.

(Selective) Copying For the (selective) copying task, none of the models that we examined, including S4D and Mamba were able to generalize to sequences with a longer random sequence part. This is not unexpected, since earlier works have already reported failure of sequential models without specific memory mechanisms in generalizing to longer sequences. This includes both studies on the copying task (Graves et al., 2014) and on a similar formal language task of string duplication (Deletang et al., 2023).

In the context of linear RNNs, it is particularly interesting to examine whether SSM models with enhanced memory components, such as xLSTM (Beck et al., 2024), would show stronger performance on memory-intensive tasks like copying. Additionally, another promising direction could involve devising regularization methods, similar to those proposed for RNNs by Wang & Niepert (2019), to guide these models towards more effective use of their memory components[8].

Counting For counting, we interestingly observe that only the S4D model generalizes from predicting one signal offset to detecting a second signal arriving immediately. In contrast to the state-tracking tasks, S4D can generalize when the temporal dynamics of the task are changed, such as increasing ITI length, unlike RNNs or Transformer models. Conversely, Mamba was unable to generalize and predict the offset, which at first glance was unexpected, as the Mamba model is theoretically an improvement over S4D. As pointed out before, Mamba has been shown to suffer from expressivity issues on tasks like parity check due to its non-negative state transition matrices; whether or not this specific parameterization could negatively affect its performance on counting tasks as well is worth exploring.

5.4 Future Research Directions

Finally, we would like to comment on a wider range of ideas relevant to our study that were not covered here as they were beyond the scope of the current work.

As alluded to earlier, there is evidence that, similar to Transformers, there does not exist a straightforward concept of state in SSMs (Merrill et al., 2024). Hence, the question of identifying states of a corresponding automaton for SSMs may not be as well-defined as it is for RNNs. However, there remain many points still under exploration in understanding SSMs, among them the interpretation of their states. Considering that state extraction can provide the most direct and straightforward way of interpreting the learning process of a model, this sounds like a very interesting direction.

With such methods devised, it becomes possible to apply state regularization techniques developed for RNNs, enhancing the utilization of their hidden states (Wang & Niepert, 2019). If similar methods could be designed

[7] Similar to (Sarrof et al., 2024), by time-invariant we mean that the transition matrix of the SSM, $\boldsymbol{A}$, is constant and hence independent of the input token.

[8] One should note that the notion of state is not as well-defined for SSMs as it is for RNNs, and consequently, its regularization is also less well-defined.

for SSMs without compromising their parallelizability, architectures with enhanced memory abilities, like xLSTM (Beck et al., 2024), could greatly benefit. This was the case for LSTM, where regularization led to significant improvements. Implementing these strategies for SSMs could potentially unlock new levels of performance and efficiency in state management.

Apart from these, extending this study to tasks that are of special interest to large language models (LLMs) is highly valuable. This includes more recent LLM-representative tasks like multi-query associative recall (MQAR), which has been shown to be strongly indicative of an LLM's performance in in-context learning. (Arora et al., 2023).

References

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, 2016.

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.

R.C. Atkinson and R.M. Shiffrin. Human memory: A proposed system and its control processes. In Kenneth W. Spence and Janet Taylor Spence (eds.), *Psychology of Learning and Motivation*, volume 2, pp. 89–195. Academic Press, 1968.

Alan D. Baddeley and Graham Hitch. Working memory. In Gordon H. Bower (ed.), *Psychology of Learning and Motivation*, volume 8, pp. 47–89. Academic Press, 1974.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.

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.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

Sarath Chandar, Chinnadhurai Sankar, Eugene Vorontsov, Samira Ebrahimi Kahou, and Yoshua Bengio. Towards non-saturating recurrent units for modelling long-term dependencies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3280–3287, 2019.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*, October 2014.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In *International Conference on Machine Learning*, 07–09 Jul 2015.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

Nelson Cowan. What are the differences between long-term, short-term, and working memory? *Progress in brain research*, 169:323–338, 2008.

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.

Gregoire Deletang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, and Pedro A Ortega. Neural networks and the chomsky hierarchy. In *International Conference on Learning Representations*, 2023.

Quentin Fournier, Gaétan Marceau Caron, and Daniel Aloise. A practical survey on faster and lighter transformers. *ACM Computing Surveys*, 55(14s):1–40, 2023.

C Randy Gallistel and John Gibbon. Time, rate, and conditioning. *Psychological review*, 107(2):289, 2000.

F.A. Gers and J. Schmidhuber. Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pp. 189–194 vol.3, 2000.

Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143, 2002.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Riccardo Grazzi, Julien Siems, Jörg K. H. Franke, Arber Zela, Frank Hutter, and Massimiliano Pontil. Unlocking state-tracking in linear rnns through negative eigenvalues, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Language Modeling*, 2024.

Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo: Recurrent memory with optimal polynomial projections. In *Conference on Advances in Neural Information Processing Systems*, 2020.

Albert Gu, Isys Johnson, Karan Goel, Khaled Kamal Saab, Tri Dao, Atri Rudra, and Christopher Re. Combining recurrent, convolutional, and continuous-time models with linear state space layers. In *Conference on Advances in Neural Information Processing Systems*, 2021.

Albert Gu, Karan Goel, Ankit Gupta, and Christopher Ré. On the parameterization and initialization of diagonal state space models. In *Conference on Advances in Neural Information Processing Systems*, 2022a.

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

Caglar Gulcehre, Sarath Chandar, and Yoshua Bengio. Memory augmented neural networks with wormhole connections, 2017.

Caglar Gulcehre, Sarath Chandar, Kyunghyun Cho, and Yoshua Bengio. Dynamic neural turing machine with continuous and discrete addressing schemes. *Neural Computation*, 30(4):857–884, 04 2018. ISSN 0899-7667.

Mikael Henaff, Arthur Szlam, and Yann LeCun. Recurrent orthogonal networks and long-memory tasks. In *International Conference on Machine Learning*, 20–22 Jun 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997. ISSN 0899-7667.

Li Jing, Caglar Gulcehre, John Peurifoy, Yichen Shen, Max Tegmark, Marin Soljacic, and Yoshua Bengio. Gated orthogonal recurrent units: On learning to forget. *Neural Computation*, 31(4):765–783, 04 2019. ISSN 0899-7667.

Michael I. Jordan. Serial order: a parallel distributed processing approach. technical report, june 1985-march 1986. Technical report, University of California, San Diego: Institute for Cognitive Science, 1986.

Najoung Kim and Sebastian Schuster. Entity tracking in language models. In *Annual Meeting of the Association for Computational Linguistics*, July 2023.

Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units, 2015.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. In *International Conference on Learning Representations*, 2023.

Robert H. Logie and Alan D. Baddeley. Cognitive processes in counting. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 13(2):310–326, April 1987. ISSN 0278-7393.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*, 11:531–545, 2023.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. In *International Conference on Machine Learning*, 2024.

George A Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review*, 63(2):81, 1956.

Marie-Pascale Noël. Counting on working memory when learning to count and to add: A preschool study. *Developmental Psychology*, 45(6):1630–1643, November 2009. ISSN 0012-1649.

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning*, 23–29 Jul 2023.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, 2013.

Zhen Qin, Songlin Yang, and Yiran Zhong. Hierarchically gated recurrent neural network for sequence modeling. In *Conference on Advances in Neural Information Processing Systems*, 2023.

Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

François Rivest, John F Kalaska, and Yoshua Bengio. Alternative time representation in dopamine models. *Journal of computational neuroscience*, 28:107–130, 2010.

David E. Rumelhart, James L. McClelland, and PDP Research Group. *Parallel Distributed Processing, Volume 1: Explorations in the Microstructure of Cognition: Foundations*. The MIT Press, 07 1986.

Yash Sarrof, Yana Veitsman, and Michael Hahn. The expressive capacity of state space models: A formal language perspective. In *Conference on Advances in Neural Information Processing Systems*, 2024.

H.T. Siegelmann and E.D. Sontag. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150, 1995. ISSN 0022-0000.

Mirac Suzgun, Yonatan Belinkov, Stuart Shieber, and Sebastian Gehrmann. LSTM networks can perform dynamic counting. In *Workshop on Deep Learning and Formal Languages: Building Bridges*, August 2019.

Corentin Tallec and Yann Ollivier. Can recurrent neural networks warp time? *arXiv preprint arXiv:1804.11188*, 2018.

Jos van der Westhuizen and Joan Lasenby. The unreasonable effectiveness of the forget gate, 2018.

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

Cheng Wang and Mathias Niepert. State-regularized recurrent neural networks, 2019.

Gail Weiss, Yoav Goldberg, and Eran Yahav. On the practical computational power of finite precision RNNs for language recognition. In *Annual Meeting of the Association for Computational Linguistics*, July 2018.

Gilad Yehudai, Haim Kaplan, Asma Ghandeharioun, Mor Geva, and Amir Globerson. When can transformers count to n?, 2024.