Exploring the Limitations of Mamba in COPY and CoT Reasoning

Ruifeng Ren[1] **Zhicong Li**[1] **Yong Liu**[1,2,3†]
[1] Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China
[2] Beijing Key Laboratory of Research on Large Models and Intelligent Governance
[3] Engineering Research Center of Next-Generation Intelligent Search and Recommendation, MOE
{renruifeng920, zhicongli, liuyonggsai}@ruc.edu.cn

Abstract

Transformers have become the backbone of modern Large Language Models (LLMs); however, their inference overhead grows linearly with the sequence length, posing challenges for modeling long sequences. In light of this, Mamba has attracted attention for maintaining a constant inference size, with empirical evidence demonstrating that it can match Transformer performance in sequence modeling while significantly reducing computational costs. However, an open question remains: *can Mamba always bring savings while achieving performance comparable to Transformers?* In this paper, we focus on analyzing the expressive ability of Mamba to perform our defined COPY operation and Chain of Thought (CoT) reasoning. First, inspired by the connection between Mamba and linear attention, we show that constant-sized Mamba may struggle to perform COPY operations while Transformers can handle them more easily. However, when the size of Mamba grows linearly with the input sequence length, it can accurately perform COPY, but in this case, Mamba no longer provides overhead savings. Based on this observation, we further analyze Mamba's ability to tackle CoT tasks, which can be described by the Dynamic Programming (DP) problems. Our findings suggest that to solve arbitrary DP problems, the total cost of Mamba is still comparable to standard Transformers. However, similar to efficient Transformers, when facing DP problems with favorable properties such as locality, Mamba can provide savings in overhead. Our experiments on the copy and CoT tasks further demonstrate Mamba's limitations compared to Transformers in learning these tasks.

1 Introduction

Reccently, Transformer-based large language models (LLMs) have become the mainstream of modern neural network architectures due to their outstanding performance across a wide range of tasks (Vaswani et al., 2017; Kenton and Toutanova, 2019; Brown et al., 2020; Dosovitskiy et al., 2020; Min et al., 2022). However, the core component of Transformers—the attention layer—while providing excellent performance, also leads to emerging drawbacks: during training, the computational cost scales quadratically with sequence length, and during inference, the cost scales linearly with sequence length. This limitation becomes increasingly unacceptable when dealing with long sequence tasks. To address this issue, many works have attempted to improve the attention mechanism to reduce its time and memory costs (Tay et al., 2023; Choromanski et al., 2020; Katharopoulos et al., 2020; Beltagy et al., 2020; Child et al., 2019). However, these improved structures often achieve efficiency in the attention layer at the expense of some performance.

Faced with the scaling challenges of Transformers, the exploration of new model architectures to replace Transformers has gradually come into focus, leading to the development of modern RNN architectures, including RWKV (Peng et al., 2023), RetNet (Sun et al., 2023), and Mamba (Gu and Dao, 2023). Among them, the Mamba architecture (Gu et al., 2021; Gu and Dao, 2023), based on the state space model (SSM), has garnered attention for its performance comparable to Transformers in many sequence modeling tasks (Dao and Gu, 2024) and vision tasks (Zhu et al., 2024; Xu et al., 2024). These models utilize hardware-aware algorithms during training, resulting in computational costs that scale linearly with sequence length, and require constant-level computation and memory during inference at each step. Mamba's strong performance and computational efficiency make it a strong competitor to Transformers.

Despite Mamba demonstrating excellent performance, one can not help but ask: Can Mamba always enjoy such "free lunch", that is, can Mamba always bring overhead savings while solving tasks

† Corresponding author

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 12551–12575
November 4-9, 2025

effectively? More recent results have revealed Mamba's shortcomings in certain tasks, especially those involving model's retrieval ability (Arora et al., 2023; Hendrycks et al., 2020; Jelassi et al., 2024). Specifically, Akyürek et al. (2024) study the in-context language learning capabilities of different models and find that Transformers outperformed other models, including Mamba, due to the specialized attention heads. Jelassi et al. (2024) also discover that Transformers are superior to Mamba on tasks that require copying from the input context. Park et al. (2024) point out that Mamba struggles to retrieve vectors from the context of multi-query associative recall (MQAR) (Arora et al., 2023), while Transformers can easily handle it well. Furthermore, Waleffe et al. (2024) conduct experiments on larger models (up to 8B parameters) with a broader range of tasks, discovering that when it comes to in-context learning and recalling information from text, although Mambas can contain the same knowledge as Transformers, it will be more difficult for them to directly copy useful information from history.

Although there has been some empirical exploration, the theoretical investigation concerning the above "free lunch" question still remains open to explore. In this paper, inspired by the comparison between Mamba and linear attention mechanism, we first focus on Mamba's ability to perform our defined COPY operation, which is closely related to the ability to retrieve information from context. Our theoretical results suggest that constant-sized Mamba may struggle with the COPY operation due to its fixed inference cost that does not scale with sequence length, whereas Transformers handle it more easily. However, if Mamba's model size scales linearly with the sequence length, it becomes capable of performing the COPY operation. Further, following the framework established by Feng et al. (2024); Yang et al. (2024), we explore Mamba's capability to reason via Chain-of-Thought (CoT), which can be formulated as dynamic programming (DP) problems. We find that to solve arbitrary DP problems, Mamba and Transformers seem to be on equal footing in terms of inference cost; however, Mamba may offer savings when dealing with m-locality DP problems like efficient Transformers (Yang et al., 2024). Our results can be concluded as follows:

- Inspired by the connection between linear attention and the SSM module, we investigate Mamba's ability to perform the COPY operation, showing that constant-sized SSM modules are less effective than attention in this task, unless the model size scales linearly with the sequence length (in Section 3);
- When equipped with CoT, the total cost required by Mamba to solve arbitrary DP problems is comparable to that of standard and efficient Transformers. However, when the DP problems have locality properties, Mamba can bring savings in overhead compared to standard Transformers (in Section 4);
- We conduct experiments on both the copy and CoT tasks, demonstrating Mamba's limitations compared to Transformers in learning these tasks (in Section 5).

2 Preliminaries

In this section, we introduce the Mamba structure that we focus on and its reformulated form firstly introduced by Han et al. (2024), which facilitates a better understanding of the connection between Mamba and linear attention.

State Space Model: The state space model (SSM) is inspired by the continuous system that maps a scalar input $x(t) \in \mathbb{R}$ to its output $y(t) \in \mathbb{R}$ through a high-dimensional hidden state $\boldsymbol{h} \in \mathbb{R}^{d_h}$ (Gu and Dao, 2023; Han et al., 2024; Zhu et al., 2024). Specifically, this system can be written as:

$$\boldsymbol{h}'(t) = \boldsymbol{A}\boldsymbol{h}(t) + \boldsymbol{b}x(t), \quad y(t) = \boldsymbol{c}^T\boldsymbol{h}(t) + dx(t),$$

where $\boldsymbol{A} \in \mathbb{R}^{d_h \times d_h}$ denotes the evolution parameters, $\boldsymbol{b}, \boldsymbol{c} \in \mathbb{R}^{d_h}$ are projection parameters and d is a scalar parameter. This continuous system can be discretized using zero-order hold (ZOH), resulting in a discrete version that can be used for neural networks. In this process, $\boldsymbol{A}, \boldsymbol{b}$ will be transformed as $\overline{\boldsymbol{A}}, \overline{\boldsymbol{b}}$. The discrete version can be written as:

$$\boldsymbol{h}_i = \overline{\boldsymbol{A}}\boldsymbol{h}_{i-1} + \overline{\boldsymbol{b}}x_i, \quad y_i = \boldsymbol{c}^T\boldsymbol{h}_i + dx_i,$$

where $\overline{\boldsymbol{A}} = \exp(\Delta\boldsymbol{A})$, $\overline{\boldsymbol{b}} = (\Delta\boldsymbol{A})^{-1}(\exp(\Delta\boldsymbol{A}) - \boldsymbol{I}) \cdot \Delta\boldsymbol{b} \approx \Delta\boldsymbol{b}$ and $\Delta \in \mathbb{R}$ is a timescale parameter. The matrix $\boldsymbol{A}$ is often assumed to be structured, e.g., diagonal, resulting in structured SSMs (Gu et al., 2022; Gupta et al., 2022).

Selective State Space Module: To enhance the SSM, Mamba makes $\boldsymbol{b}_i, \boldsymbol{c}_i, \Delta_i$ dependent on different inputs x_i. Specifically, $\boldsymbol{A}$ is set to be

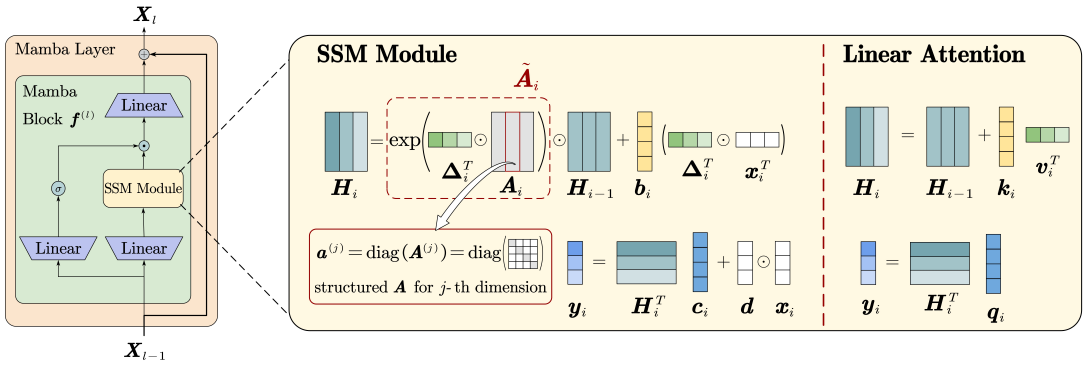

Figure 1: The illustration of the simplified Mamba layer we focus on. **Left Part**: A Mamba layer can be composed of a Mamba block with the residual connection; The Mamba block uses a gated MLP to control the output of the SSM module, where we call the branch with the SSM module as "the SSM branch" while the other as "the gated branch"; **Right Part:** The SSM module used in Mamba can be rewritten in a form similar to linear attention, where $\boldsymbol{\Delta}_i$, $\boldsymbol{b}_i$, and $\boldsymbol{c}_i$ in SSM are all derived from the current $\boldsymbol{x}_i$, similar to $\boldsymbol{v}_i$, $\boldsymbol{k}_i$, and $\boldsymbol{q}_i$ in linear attention respectively.

diagonal resulting in that $\overline{\boldsymbol{A}}_i \boldsymbol{h}_{i-1} = \tilde{\boldsymbol{a}}_i \odot \boldsymbol{h}_{i-1}$ where $\tilde{\boldsymbol{a}}_i = \exp(\Delta_i \boldsymbol{a})$, $\boldsymbol{a} = \mathrm{diag}(\boldsymbol{A})$ and $\odot$ denotes the element-wise product. In addition, $\overline{\boldsymbol{b}}_i x_i = \Delta_i \boldsymbol{b}_i x_i = \boldsymbol{b}_i(\Delta_i \odot x_i)$. Thus, this transformation ultimately results in:

$$\begin{aligned} \boldsymbol{h}_i &= \tilde{\boldsymbol{a}}_i \odot \boldsymbol{h}_{i-1} + \boldsymbol{b}_i(\Delta_i \odot x_i), \\ y_i &= \boldsymbol{c}_i^T \boldsymbol{h}_i + d \odot x_i. \end{aligned}$$

Furthermore, to extend the case of processing scalar inputs x_i to vectors $\boldsymbol{x}_i \in \mathbb{R}^d$, Mamba performs the above operations on each dimension independently, which can be formalized as:

$$\begin{aligned} \boldsymbol{H}_i &= \widetilde{\boldsymbol{A}}_i \odot \boldsymbol{H}_{i-1} + \boldsymbol{b}_i(\boldsymbol{\Delta}_i \odot \boldsymbol{x}_i)^T, \\ \boldsymbol{y}_i &= \boldsymbol{H}_i^T \boldsymbol{c}_i + \boldsymbol{d} \odot \boldsymbol{x}_i, \end{aligned} \tag{1}$$

where we have $\widetilde{\boldsymbol{A}}_i = [\tilde{\boldsymbol{a}}_i^{(j)}]_{j=1}^d \in \mathbb{R}^{d_h \times d}$, $\boldsymbol{b}_i = \boldsymbol{W}_{\boldsymbol{b}} \boldsymbol{x}_i \in \mathbb{R}^{d_h}$, $\boldsymbol{c}_i = \boldsymbol{W}_{\boldsymbol{c}} \boldsymbol{x}_i \in \mathbb{R}^{d_h}$ and $\boldsymbol{\Delta}_i = \mathrm{Softplus}(\boldsymbol{W}_{\boldsymbol{\Delta}}^2 \boldsymbol{W}_{\boldsymbol{\Delta}}^1 \boldsymbol{x}_i) \in \mathbb{R}^d$. Thus, given the input $\boldsymbol{X} = [\boldsymbol{x}_i]_{i=1}^N \in \mathbb{R}^{d \times N}$, we denote the output of the SSM module in Mamba as $\boldsymbol{Y} = \mathrm{SSM}(\boldsymbol{X})$ where $\boldsymbol{Y} = [\boldsymbol{y}_i]_{i=1}^N$ and $\boldsymbol{y}_i$ follows Eq (1). This formalization was introduced by Han et al. (2024) to build a bridge between Mamba and linear attention and here we follow this form.

Mamba Layer: Given an input sequence $\boldsymbol{X} = [\boldsymbol{x}_i]_{i=1}^N \in \mathbb{R}^{d \times N}$, it will be processed by stacked Mamba layers, each comprising a residual connection and a Mamba block $\boldsymbol{f}^{(l)} : \mathbb{R}^d \to \mathbb{R}^d$. The output of the l-th layer can be formulated as

$$\boldsymbol{X}^{l+1} = \boldsymbol{X}^l + \boldsymbol{f}^{(l)}(\boldsymbol{X}^l), \tag{2}$$

$$\boldsymbol{f}^{(l)}(\boldsymbol{X}^l) = \boldsymbol{W}_3^l \cdot \mathrm{SSM}(\boldsymbol{Z}_1^l) \odot \sigma(\boldsymbol{Z}_2^l), \tag{3}$$

where $\boldsymbol{Z}_1^l = \boldsymbol{W}_1^l \boldsymbol{X}^l + \boldsymbol{b}_1^l$, $\boldsymbol{Z}_2^l = \boldsymbol{W}_2^l \boldsymbol{X}^l + \boldsymbol{b}_2^l$ and $\sigma(\cdot)$ denotes SiLU activation function. A Mamba block includes the output of the SSM module[1]. Here we call the branch with the SSM module as "the SSM branch" and the other as "the gated branch". The model structure we consider is illustrated in Figure 1. For clarity of theoretical analysis, we retain the most essential components, namely the SSM module and the gated MLP, while slightly omitting other structures such as a causal convolution module and its subsequent non-linear activation. These omitted structures ususally account for a minor portion of the model parameters and primarily capture local dependencies, making them unlikely to significantly close the long-range modeling gap discussed in Section 3 relative to non-linear attention. Therefore, this does not affect our understanding of the overall architectural limitations. Our experiments in Section 5 also show that even when these components are included in practice, Mamba still exhibits a certain performance gap relative to Transformers on the studied tasks.

3 Can Mamba always Perform COPY Perfectly?

In this section, we begin by interpreting the reformulated SSM module as in Section 2 as a special case of linear attention, and then explore Mamba's ability to perform COPY operations during inference, which is crucial to retrieving contextual information during model's reasoning.

3.1 Viewing Mamba as linear attention

The attention mechanism is the key to the success of Transformers. Recent works has explored the

[1]To avoid confusion, we clarify that the SSM module here is specifically the Selective State Space Module used in Mamba, which is followed throughout the rest of the paper.

relationship between Mamba and attention mechanisms particularly the linear attention (Han et al., 2024; Dao and Gu, 2024; Sieber et al., 2024). The linear causal attention can be formalized as:

$$\boldsymbol{y}_i = \sum_{j=1}^{i} \boldsymbol{v}_j \boldsymbol{k}_j^T \boldsymbol{q}_i = \sum_{j=1}^{i} (\boldsymbol{q}_i^T \boldsymbol{k}_j) \boldsymbol{v}_j = \sum_{j=1}^{i} a_{ij} \boldsymbol{v}_j, \tag{4}$$

where $\boldsymbol{q}_i$, $\boldsymbol{k}_i$, $\boldsymbol{v}_i$ are usually interpreted as query, key, value respectively and a_{ij} denotes the attention scores of the i-th token to the j-th one. In attention mechanisms in Transformers, there exists $a_{ij} > 0$ for all $j \leq i$ and $\sum_{i=1}^{j} a_{ij} = 1$, which can be implemented by Softmax function.

On the other hand, given the input sequence $[\boldsymbol{x}_i]_{i=1}^N$, the output of the SSM module formulated as Eq (1) will have the following form when we set $\boldsymbol{H}_0$ and $\boldsymbol{d}$ to be zeros:

$$\boldsymbol{y}_i = (\boldsymbol{\Delta}_i \odot \boldsymbol{x}_i) \boldsymbol{b}_i^T \boldsymbol{c}_i + \sum_{j=1}^{i-1} \left[\boldsymbol{\Pi}_j \odot (\boldsymbol{\Delta}_j \odot \boldsymbol{x}_j) \boldsymbol{b}_j^T \right] \boldsymbol{c}_i, \tag{5}$$

where $\boldsymbol{\Pi}_j = \widetilde{\boldsymbol{A}}_i \odot \widetilde{\boldsymbol{A}}_{i-1} \odot \cdots \odot \widetilde{\boldsymbol{A}}_{j+1}$. We notice that since in practice all elements of $\boldsymbol{\Delta}$ are positive and $\boldsymbol{A}$ is set to be negative (Gu and Dao, 2023; Dao and Gu, 2024; Han et al., 2024), the elements of $\widetilde{\boldsymbol{A}}_i$ in Eq (1) will belong to the interval $[0, 1]$ as $\tilde{\boldsymbol{a}} = \mathrm{diag}(\exp(\Delta \boldsymbol{A}))$. To simplify our analysis, we replace the matrix $\widetilde{\boldsymbol{A}}_i$ with a forgetting coefficient $a_i \in [0, 1]$ (i.e., considering the case where all elements of $\widetilde{\boldsymbol{A}}_i$ are the same as a_i (Dao and Gu, 2024)). In fact, the subsequent analysis can be easily extended to the non-simplified case. Then, Eq (5) can be rewritten as

$$\boldsymbol{y}_i = \sum_{j=1}^{i} \alpha_j (\boldsymbol{c}_i^T \boldsymbol{b}_j)(\boldsymbol{\Delta}_j \odot \boldsymbol{x}_j), \tag{6}$$

where $\alpha_j = \Pi_{k=j+1}^{i} a_k$ for $j \leq i-1$ and $\alpha_i = 1$. We call α_j as the cumulative forgetting coefficient. In this form, we can observe that it bears similarities to linear attention without normalization in Eq (4), where $(\boldsymbol{\Delta}_j \odot \boldsymbol{x}_j)$, $\boldsymbol{b}_j$, $\boldsymbol{c}_i$ corresponds to $\boldsymbol{v}_j$, $\boldsymbol{k}_j$ and $\boldsymbol{q}_i$ respectively and $\boldsymbol{c}_i^T \boldsymbol{b}_j$ acts like attention scores a_{ij}. Considering $\alpha_{j-1} \leq \alpha_j$ and $\alpha_j \in [0, 1]$ for all $j \leq i$, the main difference is that each term in Eq (6) is weighted by a coefficient α_j to achieve the forgetting of inputs at longer distances while the attention mechanism in Transformers uses the constraints for attention scores imposed by Softmax to make sure the scaling of outputs.

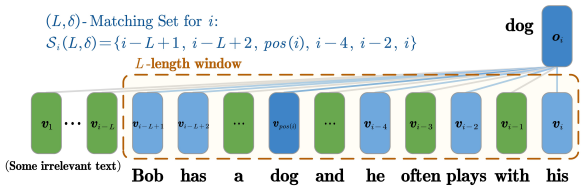

Figure 2: An example for COPY operation and (L, δ)-matching set. We expect the output at position i to be the historical record (value) corresponding to "dog". The historical records belonging to the (L, δ)-matching set are labeled in blue, which are more relevant to the output $\boldsymbol{o}_i$ based on the attention scores $|\boldsymbol{c}_i^T \boldsymbol{b}_j| \geq \delta$.

3.2 Limitation of Mamba to Perform COPY

Based on the observation of the connection between the SSM module and attention mechanism, we investigate the capability of Mamba to recover historical inputs, which is foundational for the model to process information based on context. We define the COPY operation as follows:

Definition 1 (COPY Operation)**.** *For a given SSM module and input sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$, we denote $\boldsymbol{v}_i = \boldsymbol{\Delta}_i \odot \boldsymbol{x}_i$ as historical records. Then the output of COPY operation is a sequence of vectors $\boldsymbol{o}_1, \boldsymbol{o}_2, ..., \boldsymbol{o}_N$ with $\boldsymbol{o}_i = \boldsymbol{v}_{pos(i)}$ where $pos(i)$ is the position we want to copy.*

As stated earlier, for a given SSM module, $(\boldsymbol{\Delta}_j \odot \boldsymbol{x}_j)$, $\boldsymbol{b}_j$, $\boldsymbol{c}_i$ corresponds to value, key and query respectively. Thus from the perspective of linear attention, the COPY operation for position i aims to retrieve the interested value located at $pos(i)$. Intuitively, there exists some historical records $\boldsymbol{v}_j$ that are more relevant to the current query $\boldsymbol{c}_i$ than others, which can be described as:

Definition 2 ((L, δ)-Matching Set)**.** *For a given SSM module and input sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$, the (L, δ)-matching set for $\boldsymbol{x}_i$ is defined as $\mathcal{S}_i(L, \delta) = \{j \mid |\boldsymbol{c}_i^T \boldsymbol{b}_j| \geq \delta, i - L < j \leq i\}$.*

The (L, δ)-local matching set describes the positions of historical keys $\boldsymbol{b}_j$ that is highly relevant to the current query $\boldsymbol{c}_i$ within a local window of length L, that is, the "attention scores" $|\boldsymbol{c}_i^T \boldsymbol{b}_j|$ is lower-bounded by δ. An illustration for COPY operation and (L, δ)-matching set is shown in Figure 2. We then make the following assumption:

Assumption 1. *For a given SSM module and some input sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$, the following conditions holds:*

- *There is some $L \in \mathbb{N}$ and $\delta \in \mathbb{R}$ such that for any $i \in [N]$, $S_i(L, \delta)$ exists and $pos(i) \in$*

$\mathcal{S}_i(L,\delta)$. *In addition, for any* $j \notin \mathcal{S}_i(L,\delta)$, $|\boldsymbol{c}_i^T \boldsymbol{b}_j| < \delta$.

- *For any* $i \in [N]$, $\sum_{j\in\mathcal{S}_i(L,\delta)} \alpha_j |\boldsymbol{c}_i^T \boldsymbol{b}_j| < 1$.

The first condition requires that the positions to be copied lies in $\mathcal{S}_i(L,\delta)$, which is intuitive since $\mathcal{S}_i(L,\delta)$ captures past positions most relevant to the output $\boldsymbol{o}_i$, and $pos(i)$ should be included in this set as its highly relevance. The second condition in Assumption 1 imposes a constraint on the scaling of the output caused by the records in $\mathcal{S}_i(L,\delta)$, that is, $\| \sum_{j\in\mathcal{S}_i(L,\delta)} \alpha_j \boldsymbol{c}_i^T \boldsymbol{b}_j \boldsymbol{v}_j \| < 1$ when $\|\boldsymbol{v}_j\| \leq 1$. Next, we explore the condition that can enable a given SSM module to approximate the COPY operation. Below, we provide our result:

Theorem 1 (Approximate COPY operation with constant-size SSM module)**.** *Given a SSM module with constant size and the input sequence* $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N \in [-M, M]^d$ *such that Assumption 1 holds, then for any* $\epsilon > 0$*, the SSM module can approximate COPY operation at some position* i*, that is,* $\|\boldsymbol{y}_i - \boldsymbol{o}_i\|_\infty \leq \epsilon$ *if there is* $\boldsymbol{c}_i^T \boldsymbol{b}_{pos(i)} \geq c(\frac{1}{a_{\min}})^{L-1} + d$ *where* $a_{\min} = \min_{pos(i)<j\leq i} a_j$ *and* c, d *are constants related to* ϵ, δ.

The proof can be seen in Appendix A.1. Theorem 1 shows that, relative to the sequence length N, a constant-size SSM module can approximate COPY within error ϵ if the attention score $\boldsymbol{c}_i^T \boldsymbol{b}_{pos(i)}$ is lower bounded by $c(\frac{1}{a_{\min}})^{L-1} + d$. This means that larger forgetting coefficients after $pos(i)$ help retain $\boldsymbol{v}_{pos(i)}$ to some extent, making it easier for the attention score to meet the lower bound and thereby enabling the COPY operation. Nevertheless, we note that since $a_{\min} < 1$, achieving the COPY operation requires the attention score to grow exponentially with L, where L is the distance between the current position i and the farthest highly relevant historical record. This renders the condition difficult to satisfy. However, we will show that under similar assumptions, a constant-size attention module in Transformers can perform COPY under less restrictive conditions, which can be described as follows:

Theorem 2 (Approximate COPY operation with constant-size attention module)**.** *Given a attention module with constant size and input sequence* $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N \in [-M, M]^d$ *such that Assumption 2 holds, then for any* $\epsilon > 0$*, the attention module can approximate the COPY operation at some position* i*, that is,* $\|\boldsymbol{y}_i - \boldsymbol{o}_i\|_\infty \leq \epsilon$ *if there is* $\boldsymbol{q}_i^T \boldsymbol{k}_{pos(i)} \geq \log \tilde{L} + \rho + c$ *where* $\tilde{L} = \max\{L, i - |\mathcal{S}_i(L,\delta)|\}$, $\rho = \max_{j\neq pos(i)} \boldsymbol{q}_i^T \boldsymbol{k}_j$ *and* c *is a constant related to* ϵ.

More details can be seen in Appendix A.2. It is worth noting that $\boldsymbol{q}_i^T \boldsymbol{k}_{pos(i)}$ in the attention module directly corresponds to $\boldsymbol{c}_i^T \boldsymbol{b}_{pos(i)}$ in the SSM module. Theorem 2 shows that a constant-size attention module can achieve COPY if $\boldsymbol{q}_i^T \boldsymbol{k}_{pos(i)} - \rho$ is lower bounded by $\log L$, which is much easier to satisfy than the exponential condition required for the SSM module[2]. An intuitive explanation is that despite having a constant parameter size, the attention module still maintains $O(N)$ cost when inference, allowing it sufficient capacity to store historical records and retrieve the desired one. In contrast, the SSM module with a constant size (thus $O(1)$ inference cost) can easily have the interested records overwhelmed by related but irrelevant information, making COPY more difficult to achieve. This naturally leads to the question: can we make COPY easier by increasing the size of the SSM module? In the following, we show that when its size scales linearly with the sequence length N, it is always possible for the SSM module to perform the COPY operation.

Theorem 3 (Perform COPY operation with linear-scaling size)**.** *Given the input sequence* $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N \in [-M, M]^d$*, there exists a SSM module with size* $O(N)$ *that can perform the COPY operation, that is,* $\boldsymbol{y}_i = \boldsymbol{o}_i$ *for any* $i \in [N]$.

The proof can be found in Appendix A.3. Theorem 3 is based on a simple intuition: when the model size grows linearly with the length of input sequence, the model will have enough space to store these historical records and therefore can retrieve them. However, it should be noted that such a linear-size SSM module will have the same cost as attention module in Transformers when inference, that is, $O(N)$ at each step. Thus from this perspective, Mamba does not offer savings when facing the COPY operation. Based on this observation, we will elaborate in Section 4 that when faced with CoT reasoning tasks modeled by dynamic programming problems, Mamba incurs the same order of overhead as Transformers.

4 Mamba equipped with CoT to Solve DP

Chain-of-Thought (CoT) is regarded as a powerful approach to enhancing a model's reasoning abil-

[2]As the softmax function in the attention module is shift-invariant, here we bound the attention gap $\boldsymbol{q}_i^T \boldsymbol{k}_{pos(i)} - \rho$ rather than the score $\boldsymbol{q}_i^T \boldsymbol{k}_{pos(i)}$ itself.

ity (Wei et al., 2022). It allows the model to solve complex reasoning problems by decomposing them into a sequence of simpler subproblems. During inference, the model needs to retrieve useful contexts from the reasoning chain, which is closely related to its ability to perform the COPY operation, and incrementally use these contexts to produce the final output. Such a reasoning process can be modeled as a dynamic programming (DP) problem (Feng et al., 2024; Yang et al., 2024), characterized by an input sequence, a state space, a transition function, and an aggregation function, each of which is described below.

• **Input sequences:** We use $\{s^{(1)}, s^{(2)}, \ldots, s^{(N)}\}$ to denote the input sequences and the vector $\boldsymbol{n} = \left[|s^{(1)}|, |s^{(2)}|, \ldots, |s^{(N)}|\right]^T$ to describe the size of the problem, where $|s^{(i)}|$ denotes the length of the i-th sequence.

• **State Space:** A DP problem can be decomposed into a series of sub-states to solve, forming a state space $\mathcal{I}_{\boldsymbol{n}}$ whose size depends on the problem size $\boldsymbol{n}$. Each state $i \in \mathcal{I}_{\boldsymbol{n}}$ represents an intermediate value $\mathrm{dp}(i)$ to compute, and $i \prec j$ means that state i needs to be solved before state j. The function $f_{\mathcal{I}} : \mathcal{I}_{\boldsymbol{n}} \to \mathcal{I}_{\boldsymbol{n}}$ defines the next state if j is the next state to solve after state i.

• **Transition function:** The intermediate DP values can be calculated by a transition function $f_{\mathcal{T}}$ as $\mathrm{dp}(i) = f_{\mathcal{T}}(\boldsymbol{n}, \boldsymbol{s}, \{(j, \mathrm{dp}(j)) : j \prec i\})$ where $\boldsymbol{s}$ denotes the concatenation of all tokens from the input sequences[3]. This can be rewritten as $\mathrm{dp}(i) = f_{\mathcal{T}}\left(\boldsymbol{n},\ \{\boldsymbol{s}_j : j \in \mathcal{I}_i\},\ \{\mathrm{dp}(k) : k \in \mathcal{V}_{\mathrm{dp}(i)}\}\right)$ where $\mathcal{I}_i$ and $\mathcal{V}_{\mathrm{dp}(i)}$ are the sets of input tokens indices and DP values to solve state i respectively.

• **Aggregation function:** To produce the final answer, the aggregation function collects the required intermediate DP values and calculate the final result as $A = f_{\mathcal{A}}(\{\mathrm{dp}(i) : i \in \mathcal{A}_{\boldsymbol{n}}\})$ where A denotes answer and $\mathcal{A}_{\boldsymbol{n}}$ is the set of DP values needed in the aggregation according to the problem size $\boldsymbol{n}$.

We consider how Mamba layers, as defined in Eq. (2), incrementally generate solutions to DP problems using CoT. The generated sequence follows the format:

$$s^{(1)} \mid s^{(2)} \mid \ldots \mid s^{(N)} \mid (i_1, \mathrm{dp}(i_1))\ (i_2, \mathrm{dp}(i_2))\ (i_3, \mathrm{dp}(i_3)) \ldots \left(i_{|\mathcal{I}_{\boldsymbol{n}}|}, \mathrm{dp}(i_{|\mathcal{I}_{\boldsymbol{n}}|})\right) \mid A$$

where the input sequence is separated using the symbol | as a delimiter. An classic example of DP problems is the Longest Increasing Subsequence (LIS) problem, whose goal is to find the length of the longest increasing subsequence of a given integer sequence. Following the above form, an example of the CoT output sequence can be

$$\underbrace{13\ 32\ 12\ 39\ 84}_{\text{input sequence } s^{(1)}} \mid \underbrace{1\ 2\ 1\ 3\ 4}_{\text{DP values}} \mid \underbrace{4}_{\text{final answer } A},$$

where there is only one input sequence $s^{(1)}$ for this problem. More examples for DP problems can be seen in Appendix A.4. The reasoning process of LLMs in real scenarios can generally be modeled in the form described above. Based on this formulation, we present the following result showcasing Mamba's ability to solve DP problems when equipped with CoT:

Theorem 4 (Solve DP problems with CoT)**.** *Considering any DP problem and given input sequences that satisfies Assumption 3, for any integer $T \in \mathbb{N}$, there exists several Mamba layers with size $O(T)$, such that the answer generated by the Mamba layers will be correct when the length of the answer is no more than T.*

More details can be seen in Appendix A.6. The intuition behind Theorem 4 is that when the size of the Mamba layers scales linearly with T, the model gains sufficient capacity to retrieve useful intermediate states from the reasoning chain for the next inference step, similar to the behavior described in Theorem 3. In this case, each CoT step in Mamba incurs an $O(T)$ cost, resulting in a total inference cost of $O(T^2)$. Similarly, as shown by Feng et al. (2024), a constant-sized Transformer can also solve any DP problem with CoT, but due to the per-step attention cost scaling linearly with T, the total inference cost remains $O(T^2)$. While for efficient Transformers, Yang et al. (2024) reached similar conclusions. Thus, from this prespective, Mamba also does not offer additional savings: Due to the limitation of constant inference capacity, Mamba may need to increase its model size to achieve performance comparable to that of a constant-sized Transformer, which in turn leads to higher inference cost.

It seems disappointing that, like efficient Transformers, Mamba does not reduce overhead for general DP problems. However, we argue that when DP problems exhibit favorable local properties(Yang et al., 2024), Mamba has the potential to achieve efficiency. Assuming that the output of CoT can be written as $\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_T$, if

[3]We use $s^{(i)}$ to denote the i-th input sequence while $\boldsymbol{s}_i$ to denote the i-th input token, where $\boldsymbol{s}$ is all input tokens from the concatenated input sequences.

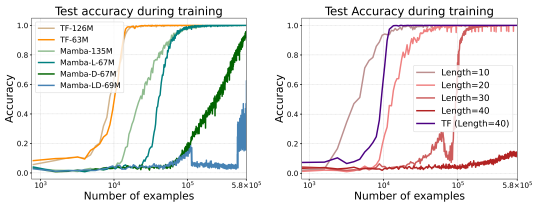

Figure 3: **Left:** Accuracy during training of models with different sizes on the copy task. **Right:** The performance of Mamba when the length of the input sequence to be copied is changed.

$\boldsymbol{o}_i = f(\{\boldsymbol{o}_j : i - m \leq j < i\})$ for any $i \in [T]$, that is, $\boldsymbol{o}_i$ only depends on at most m preceding intermediate results, then we call the DP problem is m-locality DP problem. As illustrated in Appendix A.5, the common arithmetic task can be viewed as such m-locality DP problem. Then we present the following result:

Theorem 5 (Solve m-locality DP problems with CoT)**.** *Consider any m-locality DP problem and given input sequences that satisfies Assumption 3, for any integer $T \in \mathbb{N}$, there exists several Mamba layers with size $O(m)$, such that the answer generated by the Mamba layers will be correct when the length of the answer is no more than T.*

The proof can be seen in Appendix A.7. Theorem 5 shows that when handling m-locality DP problem with CoT, the needed size of Mamba depends on the problem's locality. The cost for each step becomes a constant $O(m)$ and the total cost becomes $O(mT)$ rather than $O(T^2)$ leading to savings in cost when T is much larger than m.

5 Experimental Results

In this section, we conduct experiments to further illustrate our findings[4].

Experiments on copy tasks: We evaluate models on the copy task introduced by Jelassi et al. (2024), where the goal is to repeat an input string exactly. During training, input lengths are uniformly sampled from $[N_{\min}, N_{\max}]$, with characters drawn randomly from the alphabet. At test time, models copy strings of fixed length $N_{\max}$, and accuracy is measured by the proportion of correctly copied characters. More details are in Appendix A.8, and results are shown in Figure 3.

We first compare models of different sizes on fixed-length strings (left of Figure 3, where Transformer is denoted as TF). We find that both TF-126M and Mamba-135M eventually learn the copy task, but TF converges much faster. For smaller models, TF-63M is hardly affected even with half the layers. In contrast, smaller Mamba variants often struggle: (i) **reducing layers** (Mamba-L-67M) slows learning; (ii) **reducing the hidden size** (Mamba-D-67M) even slower; and (iii) **reducing the hidden size while increasing layers** (Mamba-LD-69M) finally fails to learn the task within finite examples and the training becomes unstable, which indicates that the hidden size has a greater impact on Mamba's performance. This confirm our findings in Section 3 that Mamba indeed finds it harder to learn the copy task compared to Transformers.

Furthermore, we change the maximum length $N_{\max}$ while maintaining the model size, as shown in the right of Figure 3. As the task becomes more challenging, Mamba requires more training examples to successfully learn the task and fails to learn within finite examples when $N_{\max} = 40$. In contrast, Transformer can still learn quickly and maintain stability even at $N_{\max} = 40$, which again indicate that Transformer outperforms Mamba in executing copy operations as in Section 3.

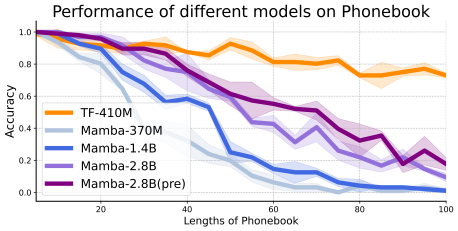

Figure 4: Comparison of different models as the Phonebook length increases.

We also conduct experiments on the phonebook task following the setting of Jelassi et al. (2024). In this task, models are firstly given a phonebook containing N names and their phone numbers such as "Bob:8651245; Alice:7656131; ...". Then in a few-shot manner, models are asked to provide the phone number for a given person in the phonebook, which relates to the ability to copy at specified positions. For Mamba, we use pretrained Mamba models with sizes of 340M, 1.4B, and 2.8B as in Gu and Dao (2023). For the Transformer baseline, we use the pretrained 410M Pythia model(Biderman et al., 2023). Additionally, we test whether providing Mamba with the task-specific target information in advance can help it better retain the targets in memory (labeled as "Mamba-2.8B(pre)"). For example, we prepend the prompt "Please remember the phone number of Bob" before presenting the

[4]Our code is available at https://github.com/Miao-Mouse/mamba-cot

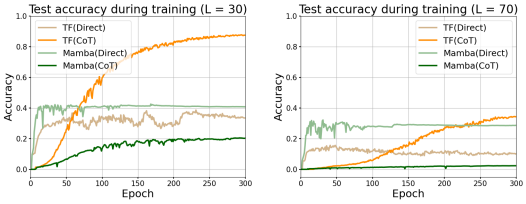

Figure 5: Accuracy during training when the task length $L = 30/70$ and $d = 256$ (TF denotes Transformer).

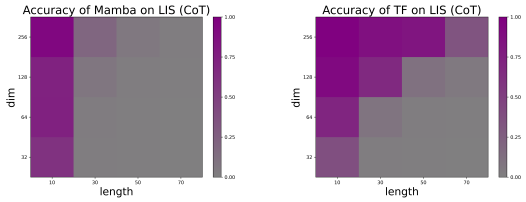

Figure 6: Accuracy of Mamba and Transformer under different task lengths and model sizes.

phonebook to encourage Mamba to consistently retain Bob's number in the subsequent sequence. All results are shown in Figure 4. It can be seen that as size increases, all Mamba models degrade quickly. When given the target information in advance, Mamba-2.8B(pre) can partially filter irrelevant content and slow this decline. However, even then, its accuracy still lags behind that of Pythia-410M on larger problems, which further illustrates its limitations compared to Transformers.

Experiments on CoT Task: In addition, we further evaluate model performance on solving DP problems, focusing on the classic Longest Increasing Subsequence (LIS) problem following the setup of Feng et al. (2024). We examined two scenarios: (i) **Direct:** the model is trained to directly output the final answer; (ii) **CoT:** the model is trained to output the answer through CoT reasoning, where it is required to generate both the correct reasoning steps and the final answer. We investigate different task lengths L (problem size) and model dimensions d. For Mamba, we adjust the number of layers to match or slightly exceed the size of the Transformer within the same d, with roughly two Mamba layers corresponding to one Transformer layer. More details can be seen in Appendix A.8.

First, we present the accuracy curves during training in Figure 5. Under the Direct setting, Mamba outperforms Transformer due to the relatively short task length. However, under the CoT setting where significantly longer sequences are required, Transformer consistently performs better. Notably, unlike Transformers, Mamba performs even worse under the CoT setting than in the direct setting. This is because Mamba's constant-sized inference capacity limits its ability to handle long reasoning chains, whereas Transformers—with overhead that grows with sequence length—are better equipped to process such information.

Furthermore, in Figure 6, we present results of Mamba and Transformer under different CoT settings. It can be seen that when the two models have comparable sizes, Mamba consistently underperforms Transformer, which supports our analysis in Section 4: while a constant-sized Transformer can effectively solve DP problems with CoT, Mamba with comparable size may struggle to do so unless given greater capacity. These findings indicate that Mamba can not always get a free lunch: the inference cost that does not scale with sequence length may limit Mamba's ability to solve certain tasks. In addition, in Appendix A.9, we show that appropriately introducing Transformer layers into Mamba can partially close the gap between the two models, which is a direction worth exploring for Mamba's future improvement.

6 Related work

SSMs and Attention Mechanism: The attention mechanism is a core component of LLMs (Vaswani et al., 2017). Drawing connections between SSMs and attention is a fascinating direction as it not only aids in our understanding of the Mamba structure but also facilitates the transfer of well-established acceleration techniques from attention mechanisms to Mamba (Yang et al., 2023; Dao and Gu, 2024; Han et al., 2024; Sieber et al., 2024). Based on observations of the similarities between them, Dao and Gu (2024) proposed the state space dual (SSD) layer based on SSMs to achieve significant improvements in training efficiency. Particularly, Han et al. (2024) reformulate the structure of SSMs to establish links with linear attention, aiming to find the key factors behind success in vision tasks. We follow this convenient reformulation to explore Mamba's ability to perform the COPY operation.

Comparisons between Transformers and Mamba: More recent works compare the performance of Mamba and Transformers across various tasks from experimental and theoretical perspectives. Jelassi et al. (2024) find that Transformers significantly surpass SSMs when facing tasks related to copying and retrieving information from context. In addition, Park et al. (2024) investigate Mamba's capability for in-context learning

and demonstrate that Mamba outperforms Transformers in sparse parity learning while it is weaker in tasks involving non-standard retrieval functionality. Similarly, Waleffe et al. (2024) conduct experiments on a larger scale and find that Mamba lag behind Transformers in tasks that require strong copying and long-context reasoning. Our experiments reference the setups of these works and conduct similar investigations. From the theoretical perspective, Merrill et al. (2024) demonstrate that similar to Transformers, Mamba is also unable to solve state tracking problems such as permutation composition while we focus on the different task of performing COPY operation. Jelassi et al. (2024) investigate the ability of generalized SSMs to replicate entire sequences that satisfy some distribution and providing a lower bound for their state space memory whereas our work focuses on analyzing the impact of the distance of the specified token to be copied on the output error from the numerical approximation perspective in Theorem 1. Additionally, Arora et al. (2023) use communication complexity to show that recurrent models require at least $\Omega(N)$ to solve Multi-Query Associative Recall (MQAR) tasks, which is a lower bound guarantee. In contrast, we provided an upper bound on the model size required for Mamba to achieve COPY using a constructive approach in our Theorem 3.

Transformers and modern RNNs with CoT: Chain-of-Thought (CoT) (Wei et al., 2022) is employed to enhance the performance of LLMs by enabling them to provide step-by-step reasoning before arriving at a final answer. It has been shown theoretically that Transformers with CoT exhibit significantly improved expressive power, allowing them to solve more complex problems compared to Transformers without CoT (Merrill and Sabharwal, 2023b; Feng et al., 2024; Merrill and Sabharwal, 2023a; Li et al., 2024; Yang et al., 2024). Our analysis of Mamba equipped with CoT follows the framework set by Feng et al. (2024); Yang et al. (2024) in their analysis of dynamic programming (DP) problems. Additionally, Wen et al. (2024) use communication complexity to show that even with CoT, any RNN model with $o(n)$ bit memory cannot solve tasks in $T \in \{$Index, AR, c-gram retrieval, Counting$\}$ of size n for large enough n, which means that the lower bound for RNNs to solve these tasks is $w(n)$. Different from this, our work explores the ability of Mamba equipped with CoT from the perspective of solving DP problems following the setting of Feng et al. (2024) and show constructions for Mamba layers with linear-scaling size relative to the sequence length to solve DP problems, which can be seen as an upper bound of the model size required to solve DP.

Limitations

In this paper, inspired by the similarity between SSM and linear attention, we explore Mamba's ability to perform the COPY operation and CoT tasks. Our findings contribute to a deeper understanding of Mamba. However, we would like to illustrate that while Mamba may slightly underperform Transformers in certain tasks, it offers advantages in others like sparse parity learning (Park et al., 2024) and can achieve comparable performance with lower costs (Gu and Dao, 2023). The theoretical mechanisms behind these advantages remain to be further explored. Additionally, as shown in Park et al. (2024); Waleffe et al. (2024); Wen et al. (2024), exploring hybrid architectures that combine the strengths of Mamba and Transformers also merits further investigation. We leave these aspects for future work.

Acknowledgments

We thank all anonymous reviewers for their constructive suggestions to improve the quality of this paper. This research was supported by National Key Research and Development Program of China (NO. 2024YFE0203200), National Natural Science Foundation of China (No.62476277), CCF-ALIMAMA TECH Kangaroo Fund(No.CCF-ALIMAMA OF 2024008), and Huawei-Renmin University joint program on Information Retrieval. We also acknowledge the support provided by the fund for building worldclass universities (disciplines) of Renmin University of China and by the funds from Beijing Key Laboratory of Big Data Management and Analysis Methods, Gaoling School of Artificial Intelligence, Renmin University of China, from Engineering Research Center of Next-Generation Intelligent Search and Recommendation, Ministry of Education, from Intelligent Social Governance Interdisciplinary Platform, Major Innovation & Planning Interdisciplinary Platform for the "DoubleFirst Class" Initiative, Renmin University of China, from Public Policy and Decision-making Research Lab of Renmin University of China, and from Public Computing Cloud, Renmin University of China.

References

Ekin Akyürek, Bailin Wang, Yoon Kim, and Jacob Andreas. 2024. In-context language learning: Architectures and algorithms. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 787–812. PMLR.

Alex Andonian, Quentin Anthony, Stella Biderman, Sid Black, Preetham Gali, Leo Gao, Eric Hallahan, Josh Levy-Kramer, Connor Leahy, Lucas Nestler, Kip Parker, Michael Pieler, Jason Phang, Shivanshu Purohit, Hailey Schoelkopf, Dashiell Stander, Tri Songz, Curt Tigges, Benjamin Thérien, and 2 others. 2023. GPT-NeoX: Large Scale Autoregressive Language Modeling in PyTorch.

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

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, and 1 others. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, and 1 others. 2020. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*.

Tri Dao and Albert Gu. 2024. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, and 1 others. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. 2024. Towards revealing the mystery behind chain of thought: a theoretical perspective. *Advances in Neural Information Processing Systems*, 36.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, and 1 others. 2020. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Albert Gu and Tri Dao. 2023. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Albert Gu, Karan Goel, Ankit Gupta, and Christopher Ré. 2022. On the parameterization and initialization of diagonal state space models. *Advances in Neural Information Processing Systems*, 35:35971–35983.

Albert Gu, Karan Goel, and Christopher Ré. 2021. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*.

Ankit Gupta, Albert Gu, and Jonathan Berant. 2022. Diagonal state spaces are as effective as structured state spaces. *Advances in Neural Information Processing Systems*, 35:22982–22994.

Dongchen Han, Ziyi Wang, Zhuofan Xia, Yizeng Han, Yifan Pu, Chunjiang Ge, Jun Song, Shiji Song, Bo Zheng, and Gao Huang. 2024. Demystify mamba in vision: A linear attention perspective. *arXiv preprint arXiv:2405.16605*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Samy Jelassi, David Brandfonbrener, Sham M Kakade, and Eran Malach. 2024. Repeat after me: Transformers are better than state space models at copying. *arXiv preprint arXiv:2402.01032*.

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

Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of naacL-HLT*, volume 1, page 2.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. 2024. Chain of thought empowers transformers to solve inherently serial problems. *arXiv preprint arXiv:2402.12875*.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, and 1 others. 2024. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*.

I Loshchilov. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

William Merrill, Jackson Petty, and Ashish Sabharwal. 2024. The illusion of state in state-space models. *arXiv preprint arXiv:2404.08819*.

William Merrill and Ashish Sabharwal. 2023a. The expresssive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*.

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

Erxue Min, Runfa Chen, Yatao Bian, Tingyang Xu, Kangfei Zhao, Wenbing Huang, Peilin Zhao, Junzhou Huang, Sophia Ananiadou, and Yu Rong. 2022. Transformer for graphs: An overview from architecture perspective. *arXiv preprint arXiv:2202.08455*.

Jongho Park, Jaeseung Park, Zheyang Xiong, Nayoung Lee, Jaewoong Cho, Samet Oymak, Kangwook Lee, and Dimitris Papailiopoulos. 2024. Can mamba learn how to learn? A comparative study on in-context learning tasks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 39793–39812. PMLR.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, and 1 others. 2023. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*.

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

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

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. 2023. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2023. Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6):109:1–109:28.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Roger Waleffe, Wonmin Byeon, Duncan Riach, Brandon Norick, Vijay Korthikanti, Tri Dao, Albert Gu, Ali Hatamizadeh, Sudhakar Singh, Deepak Narayanan, and 1 others. 2024. An empirical study of mamba-based language models. *arXiv preprint arXiv:2406.07887*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Kaiyue Wen, Xingyu Dang, and Kaifeng Lyu. 2024. Rnns are not transformers (yet): The key bottleneck on in-context retrieval. *arXiv preprint arXiv:2402.18510*.

Rui Xu, Shu Yang, Yihui Wang, Bo Du, and Hao Chen. 2024. A survey on vision mamba: Models, applications and challenges. *arXiv preprint arXiv:2404.18861*.

Kai Yang, Jan Ackermann, Zhenyu He, Guhao Feng, Bohang Zhang, Yunzhen Feng, Qiwei Ye, Di He, and Liwei Wang. 2024. Do efficient transformers really save computation? *arXiv preprint arXiv:2402.13934*.

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

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

A Appendix

A.1 Proof of Theorem 1

Theorem 6 (Approximate COPY operation with constant-size SSM module)**.** *Given a SSM module with constant size and input sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N \in [-M, M]^d$ such that Assumption 1 holds, then for any $\epsilon > 0$, the SSM module can approximate the COPY operation at some position i, that is, $\|\boldsymbol{y}_i - \boldsymbol{o}_i\|_\infty \leq \epsilon$ if there is $\boldsymbol{c}_i^T \boldsymbol{b}_{pos(i)} \geq c(\frac{1}{a_{\min}})^{L-1} + d$ where $a_{\min} = \min_{pos(i)<j\leq i} a_j$ and c, d are constants related to ϵ, δ.*

Proof. We firstly show that given the i-th input $\boldsymbol{x}_i$, a SSM module can retrieve the most relevant historical record $\boldsymbol{v}_{pos(i)}$ from the hidden state and perform the defined COPY operation under the condition illustrated in our theorem. To achieve this, recalling that $\boldsymbol{v}_j = \boldsymbol{\Delta}_j \odot \boldsymbol{x}_j$ in Eq (6) and denoting $\|\boldsymbol{\Delta}\|_\infty = \max_{i\in[N]} \|\boldsymbol{\Delta}_i\|_\infty$, we have that

$$\begin{aligned}
\left\|\boldsymbol{y}_i - \boldsymbol{v}_{pos(i)}\right\|_\infty &= \left\|\sum_{j=1}^{i} \alpha_j (\boldsymbol{\Delta}_j \odot \boldsymbol{x}_j)\boldsymbol{b}_j^T \boldsymbol{c}_i - \boldsymbol{v}_{pos(i)}\right\|_\infty = \left\|\sum_{j=1}^{i} \alpha_j (\boldsymbol{c}_i^T \boldsymbol{b}_j)\boldsymbol{v}_j - \boldsymbol{v}_{pos(i)}\right\|_\infty && (7)\\
&= \left\|\sum_{j\neq pos(i)} \alpha_j (\boldsymbol{c}_i^T \boldsymbol{b}_j)\boldsymbol{v}_j + \alpha_{pos(i)}(\boldsymbol{c}_i^T \boldsymbol{b}_{pos(i)})\boldsymbol{v}_{pos(i)} - \boldsymbol{v}_{pos(i)}\right\|_\infty && (8)\\
&= \left\|\sum_{j\neq pos(i)} \alpha_j (\boldsymbol{c}_i^T \boldsymbol{b}_j)\boldsymbol{v}_j + \left[\alpha_{pos(i)}(\boldsymbol{c}_i^T \boldsymbol{b}_{pos(i)}) - 1\right]\boldsymbol{v}_{pos(i)}\right\|_\infty && (9)\\
&\leq M\|\boldsymbol{\Delta}\|_\infty \left(\sum_{j\neq pos(i)} \alpha_j |\boldsymbol{c}_i^T \boldsymbol{b}_j| + 1 - \alpha_{pos(i)} \boldsymbol{c}_i^T \boldsymbol{b}_{pos(i)}\right), && (10)
\end{aligned}$$

where in (10) we use the fact that $\|\boldsymbol{x}_i\|_\infty \leq M$ and $\alpha_{pos(i)}|\boldsymbol{c}_i^T \boldsymbol{b}_{pos(i)}| \leq 1$ (from the second condition in Assumption 1). Thus, to prove $\left\|\boldsymbol{y}_i - \boldsymbol{v}_{pos(i)}\right\|_\infty \leq \epsilon$, we can show that

$$\boldsymbol{c}_i^T \boldsymbol{b}_{pos(i)} \geq \left(1 - \frac{\epsilon}{M\|\boldsymbol{\Delta}\|_\infty}\right)\frac{1}{\alpha_{pos(i)}} + \sum_{j\neq pos(i)} \frac{\alpha_j}{\alpha_{pos(i)}} \left|\boldsymbol{c}_i^T \boldsymbol{b}_j\right|. \tag{11}$$

Recalling that $\alpha_j = \prod_{k=j+1}^{i} a_k$, we have

$$\frac{\alpha_j}{\alpha_{pos(i)}} = \begin{cases} \prod_{k=j+1}^{pos(i)} a_k = a_{pos(i)} a_{pos(i)-1} \ldots a_{j+1}, & \text{when } j < pos(i) \\ \prod_{k=pos(i)+1}^{j} \frac{1}{a_k} = \frac{1}{a_j a_{j-1} \ldots a_{pos(i)+1}}, & \text{when } j > pos(i). \end{cases} \tag{12}$$

Then we can consider the second term on the right side of Inequality (11) as

$$\sum_{j\neq pos(i)} \frac{\alpha_j}{\alpha_{pos(i)}} \left|\boldsymbol{c}_i^T \boldsymbol{b}_j\right| = \sum_{j\notin S_i} \frac{\alpha_j}{\alpha_{pos(i)}} \left|\boldsymbol{c}_i^T \boldsymbol{b}_j\right| + \sum_{j\in S_i, j\neq pos(i)} \frac{\alpha_j}{\alpha_{pos(i)}} \left|\boldsymbol{c}_i^T \boldsymbol{b}_j\right| \tag{13}$$

For the first term on the right side, we have

$$\sum_{j\notin S_i}\frac{\alpha_j}{\alpha_{pos(i)}}\left|\boldsymbol{c}_i^T\boldsymbol{b}_j\right| \le \delta\left(\sum_{j\notin S_i, j<pos(i)}\frac{\alpha_j}{\alpha_{pos(i)}}+\sum_{j\notin S_i, j>pos(i)}\frac{\alpha_j}{\alpha_{pos(i)}}\right) \tag{14}$$

$$\le \delta\left(\sum_{j=1}^{pos(i)-1}\frac{\alpha_j}{\alpha_{pos(i)}}+\sum_{j=pos(i)+1}^{i}\frac{\alpha_j}{\alpha_{pos(i)}}\right) \tag{15}$$

$$\le \delta\left(\sum_{k=1}^{pos(i)-1}(a_{\max})^k+\sum_{k=1}^{i-pos(i)}\left(\frac{1}{a_{\min}}\right)^k\right) \tag{16}$$

$$\le \delta\left(\frac{a_{\max}(1-a_{\max}^{pos(i)-1})}{1-a_{\max}}+\frac{\left(\frac{1}{a_{\min}}\right)^{i-pos(i)}-1}{1-a_{\min}}\right) \tag{17}$$

where $a_{\max}=\max_{1\le j<pos(i)}a_j$ and $a_{\min}=\min_{pos(i)<j\le i}a_j$. In (14) we use the assumption that $\left|\boldsymbol{c}_i^T\boldsymbol{b}_j\right|le\delta$ for $j\notin S_i$; in (15) we use $\frac{\alpha_j}{\alpha_{pos(i)}}>0$ for all $j\le i$; in (16), we use the fact that $\frac{\alpha_j}{\alpha_{pos(i)}}\le(a_{\max})^{pos(i)-j}$ for $j<pos(i)$ and $\frac{\alpha_j}{\alpha_{pos(i)}}\le\left(\frac{1}{a_{\min}}\right)^{j-pos(i)}$ for $j>pos(i)$; in (17), we use the formula for the sum of a geometric series.

Furthermore, considering that the vector $\boldsymbol{v}_{pos(i)}$ to be copied must exist in the L-local matching set S_i so there is $i-L+1\le pos(i)\le i$, we have the following

$$\sum_{j\notin S_i}\frac{\alpha_j}{\alpha_{pos(i)}}\left|\boldsymbol{c}_i^T\boldsymbol{b}_j\right| \le \delta\left(\frac{a_{\max}(1-a_{\max}^{pos(i)-1})}{1-a_{\max}}+\frac{\left(\frac{1}{a_{\min}}\right)^{L-1}-1}{1-a_{\min}}\right) \tag{18}$$

$$\le \delta\left(\frac{a_{\max}}{1-a_{\max}}+\frac{\left(\frac{1}{a_{\min}}\right)^{L-1}}{1-a_{\min}}\right). \tag{19}$$

In (18), we use the fact that $a_{\max}\in(0,1)$ while $\frac{1}{a_{\min}}>1$; in (19) we just ignore the term $a_{\max}^{pos(i)-1}$ for simplicity (in fact, we can find that when i is sufficiently large, the effect of this term can be neglected).

Meanwhile, with Assumption 1, we can consider the second term on the right side of Inequality (13) as

$$\sum_{j\in S_i, j\neq pos(i)}\frac{\alpha_j}{\alpha_{pos(i)}}\left|\boldsymbol{c}_i^T\boldsymbol{b}_j\right| \le \frac{1}{\alpha_{pos(i)}}-\boldsymbol{c}_i^T\boldsymbol{b}_{pos(i)}. \tag{20}$$

Thus, considering (11), (19) and (20), to show $\|\boldsymbol{y}_i-\boldsymbol{o}_i\|_\infty\le\epsilon$, the following condition should be satisfied

$$\boldsymbol{c}_i^T\boldsymbol{b}_{pos(i)} \ge \left(1-\frac{\epsilon}{M\|\boldsymbol{\Delta}\|_\infty}\right)\frac{1}{\alpha_{pos(i)}}+\delta\left(\frac{a_{\max}}{1-a_{\max}}+\frac{\left(\frac{1}{a_{\min}}\right)^{L-1}}{1-a_{\min}}\right) \tag{21}$$

$$+\frac{1}{\alpha_{pos(i)}}-\boldsymbol{c}_i^T\boldsymbol{b}_{pos(i)}. \tag{22}$$

After reformulating, there exists

$$\boldsymbol{c}_i^T \boldsymbol{b}_{pos(i)} \geq \left(1 - \frac{\epsilon}{2M\|\boldsymbol{\Delta}\|_\infty}\right) \frac{1}{\alpha_{pos(i)}} + \frac{\delta}{2} \left(\frac{a_{\max}}{1 - a_{\max}} + \frac{\left(\frac{1}{a_{\min}}\right)^{L-1}}{1 - a_{\min}} \right) \tag{23}$$

$$= \frac{\delta}{2(1 - a_{\min})} \left(\frac{1}{a_{\min}}\right)^{L-1} + \left(1 - \frac{\epsilon}{2M\|\boldsymbol{\Delta}\|_\infty}\right) \alpha_{pos(i)}^{-1} + \frac{\delta a_{\max}}{2(1 - a_{\max})} \tag{24}$$

$$= c \left(\frac{1}{a_{\min}}\right)^{L-1} + d, \tag{25}$$

where we let $c = \frac{\delta}{2(1-a_{\min})}$ and use d to denote the remaining terms. Thus, we complete our proof. □

A.2 Proof of Theorem 2

To provide the theorem, we first give the definitions and assumptions for the attention module in Transformers, which are really similar to those for the SSM module from the perspective of linear attention. For some input sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$, the output of the attention module we consider can be formulated as:

$$\boldsymbol{y}_i = \sum_{j \leq i} a_{i,j} \boldsymbol{v}_j = \sum_{j \leq i} \frac{e^{\boldsymbol{q}_i^T \boldsymbol{k}_j}}{\sum_{t \leq i} e^{\boldsymbol{q}_i^T \boldsymbol{k}_t}} \boldsymbol{v}_j, \quad i = 1, 2, \ldots, N$$

where $a_{i,j}$ the attention scores calculated by the $\mathrm{Softmax}(\cdot)$ function and $\boldsymbol{q}_i = \boldsymbol{W}_q \boldsymbol{x}_i$, $\boldsymbol{k}_i = \boldsymbol{W}_k \boldsymbol{x}_i$, $\boldsymbol{v}_i = \boldsymbol{W}_v \boldsymbol{x}_i$ are the query, key and value vectors respectively. Then, we give the definition of COPY Operation for attention module.

Definition 3 (COPY Operation for the attention module)**.** *For a given attention module and input sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$, the output of COPY operation is a sequence of vectors $\boldsymbol{o}_1, \boldsymbol{o}_2, ..., \boldsymbol{o}_N$ with $\boldsymbol{o}_i = \boldsymbol{v}_{pos(i)}$ where $pos(i) \in [N]$ is the position we want to copy.*

In fact, this definition is very similar to the COPY for the SSM module, except that the vector being copied is changed from the historical records $\boldsymbol{v}_i = \boldsymbol{\Delta}_i \odot \boldsymbol{x}_i$ in the SSM module to the value vectors in the attention module directly. From the perspective of linear attention, the former is precisely the value vectors in attention, so the two definitions are closely related: Similarly, we define the definition of L, δ)-Matching Set for the attention module:

Definition 4 ((L, δ)-Matching Set for the attention module)**.** *For a given attention module and input sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$, the (L, δ)-matching set for $\boldsymbol{x}_i$ is defined as $\mathcal{S}_i(L, \delta) = \{j \mid |\boldsymbol{q}_i^T \boldsymbol{k}_j| \geq \delta, i - L < j \leq i\}$.*

Next, we make the following assumption:

Assumption 2. *For a given attention module and input sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$, there exist some $L \in \mathbb{N}$ and $\delta \in \mathbb{R}$ such that for any $i \in [N]$, $S_i(L, \delta)$ exists and $pos(i) \in \mathcal{S}_i(L, \delta)$. In addition, for any $j \notin \mathcal{S}_i(L, \delta)$, $|\boldsymbol{q}_i^T \boldsymbol{k}_j| < \delta$.*

Note that, since the attention scores in Transformers are naturally normalized by Softmax function such that $\sum_{j \in \mathcal{S}_i} a_{i,j} \leq 1$, we ignore the second condition as in Assumption 1 here, which would impose constraints on the stability of the output. Then, we give our theorem as following

Theorem 7 (Approximate COPY operation with constant-size attention module)**.** *Given a attention module with constant size and input sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N \in [-M, M]^d$ such that Assumption 2 holds, then for any $\epsilon > 0$, the attention module can approximate the COPY operation at some position i, that is, $\|\boldsymbol{y}_i - \boldsymbol{o}_i\|_\infty \leq \epsilon$ if there is $\boldsymbol{q}_i^T \boldsymbol{k}_{pos(i)} \geq \log \tilde{L} + \rho + c$ where $\tilde{L} = \max\{L, i - |\mathcal{S}_i(L, \delta)|\}$, $\rho = \max_{j \neq pos(i)} \boldsymbol{q}_i^T \boldsymbol{k}_j$, and c is a constant related to ϵ.*

Proof. For some given i-th input $\boldsymbol{x}_i$, we have that

$$\left\|\boldsymbol{y}_i - \boldsymbol{v}_{pos(i)}\right\|_\infty = \left\|\sum_{j\le i} a_{i,j}\boldsymbol{v}_i - \boldsymbol{v}_{pos(i)}\right\|_\infty = \left\|\sum_{j\le i, j\neq pos(i)} a_{i,j}\boldsymbol{v}_i - \left(1 - a_{i,pos(i)}\right)\boldsymbol{v}_{pos(i)})\right\|_\infty \tag{26}$$

$$\le \left(1 - a_{i,pos(i)} + \sum_{j\le i} a_{i,j}\right) M\left\|\boldsymbol{W}_V\right\|_\infty = 2\left(1 - a_{i,pos(i)}\right) M\left\|\boldsymbol{W}_V\right\|_\infty. \tag{27}$$

To achieve $\|\boldsymbol{y}_i - \boldsymbol{o}_i\| \le \epsilon$, we can show that

$$a_{i,pos(i)} = \frac{e^{\boldsymbol{q}_i^T\boldsymbol{k}_{pos(i)}}}{\sum_{j\neq pos(i)} e^{\boldsymbol{q}_i^T\boldsymbol{k}_j} + e^{\boldsymbol{q}_i^T\boldsymbol{k}_{pos(i)}}} \ge 1 - \frac{\epsilon}{2M\left\|\boldsymbol{W}_V\right\|_\infty}. \tag{28}$$

Here, we omit the condition $j \le i$ for simplicity of notation and this is equivalent to show that

$$\sum_{j\neq pos(i)} e^{\boldsymbol{q}_i^T\boldsymbol{k}_j - \boldsymbol{q}_i^T\boldsymbol{k}_{pos(i)}} + 1 \le \frac{2M\left\|\boldsymbol{W}_V\right\|_\infty}{2M\left\|\boldsymbol{W}_V\right\|_\infty - \epsilon}, \tag{29}$$

$$e^{-\boldsymbol{q}_i^T\boldsymbol{k}_{pos(i)}} \sum_{j\neq pos(i)} e^{\boldsymbol{q}_i^T\boldsymbol{k}_j} \le \frac{\epsilon}{2M\left\|\boldsymbol{W}_V\right\|_\infty - \epsilon}. \tag{30}$$

Thus, we need to show that

$$e^{\boldsymbol{q}_i^T\boldsymbol{k}_{pos(i)}} \ge \frac{2M\left\|\boldsymbol{W}_V\right\|_\infty - \epsilon}{\epsilon} \sum_{j\neq pos(i)} e^{\boldsymbol{q}_i^T\boldsymbol{k}_j}. \tag{31}$$

Further, according to whether indices $j \in \mathcal{S}_i$ or not, we have

$$\boldsymbol{q}_i^T\boldsymbol{k}_{pos(i)} \ge \log\left[\frac{2M\left\|\boldsymbol{W}_V\right\|_\infty - \epsilon}{\epsilon}\left(\sum_{j\in\mathcal{S}_i, j\neq pos(i)} e^{\boldsymbol{q}_i^T\boldsymbol{k}_j} + \sum_{j\notin\mathcal{S}_i} e^{\boldsymbol{q}_i^T\boldsymbol{k}_j}\right)\right], \tag{32}$$

where we use $\mathcal{S}_i$ to denote $\mathcal{S}_i(L,\delta)$ for simplicity of notation. Then we just need to show that $\boldsymbol{q}_i^T\boldsymbol{k}_{pos(i)}$ is larger than the upper bound of the right side, that is,

$$\boldsymbol{q}_i^T\boldsymbol{k}_{pos(i)} \ge \log\left[\frac{2M\left\|\boldsymbol{W}_V\right\|_\infty}{\epsilon}\left(Le^\rho + (i - |\mathcal{S}_i|)e^\delta\right)\right], \tag{33}$$

where $\rho = \max_{j\in\mathcal{S}_i, j\neq pos(i)} \boldsymbol{q}_i^T\boldsymbol{k}_j$. Here we use the fact that there are at most L terms in the matching set $\mathcal{S}_i$ thus $\sum_{j\in\mathcal{S}_i, j\neq pos(i)} e^{\boldsymbol{q}_i^T\boldsymbol{k}_j} \le Le^\rho$ and we also use the assumption that $\boldsymbol{q}_i^T\boldsymbol{k}_j \le \delta$ for $j \notin \mathcal{S}_i$ thus we have $\sum_{j\notin\mathcal{S}_i} e^{\boldsymbol{q}_i^T\boldsymbol{k}_j} \le (i - |\mathcal{S}_i|)e^\delta$. By denoting $\tilde{L} = \max\{L, \mathcal{S}_i\}$, we can show that,

$$\boldsymbol{q}_i^T\boldsymbol{k}_{pos(i)} \ge \log\tilde{L} + \log\frac{2}{\epsilon}(e^\delta + e^\rho)M\left\|\boldsymbol{W}_V\right\|_\infty. \tag{34}$$

Noting that there is $\rho \ge \delta$ according to Assumption 2, we also have $\rho = \max_{j\neq pos(i)} \boldsymbol{q}_i^T\boldsymbol{k}_j$ and we can show that

$$\boldsymbol{q}_i^T\boldsymbol{k}_{pos(i)} \ge \log\tilde{L} + \log\frac{4}{\epsilon}e^\rho M\left\|\boldsymbol{W}_V\right\|_\infty = \log\tilde{L} + \rho + \log\frac{4}{\epsilon}M\left\|\boldsymbol{W}_V\right\|_\infty. \tag{35}$$

Thus we complete our proof. □

A.3 Proof of Theorem 3

Theorem 8 (Perform COPY operation with linear-scaling size)**.** *Given sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N \in [-M, M]^d$, there exists a SSM module with size $O(N)$ that can perform the defined COPY operation, that is, $\boldsymbol{y}_i = \boldsymbol{o}_i$ for any $i \in [N]$.*

Proof. Recalling that the output of SSM module in Mamba can be rewritten in the form of Eq (6), where $(\boldsymbol{\Delta}_j \odot \boldsymbol{x}_j), \boldsymbol{b}_j, \boldsymbol{c}_i$ corresponds to $\boldsymbol{v}_j, \boldsymbol{k}_j$ and $\boldsymbol{q}_i$ respectively. Our intuition is to store all the information of $\boldsymbol{v}_i$ from our history in the hidden state space of size $O(N)$ (similar to the KV cache in attention format), and then use the appropriate $\boldsymbol{c}_i$ as the query for retrieval. We can set $\boldsymbol{A} = \boldsymbol{O}$ so that Eq (6) further transforms in a way that does not forget historical information, that is, $\boldsymbol{y}_i = \sum_{j=1}^{i} (\boldsymbol{\Delta}_j \odot \boldsymbol{x}_j) \boldsymbol{b}_j^T \boldsymbol{c}_i = \sum_{j=1}^{i} \boldsymbol{v}_j \boldsymbol{b}_j^T \boldsymbol{c}_i$.

Let $\tilde{\boldsymbol{x}}_i = [\boldsymbol{x}_i, \boldsymbol{e}_i, \boldsymbol{e}_{pos(i)}] \in \mathbb{R}^{d+2N}$ where $\boldsymbol{e}_i \in \mathbb{R}^N$ denote the one-hot vector where only the i-th value is 1. We use $\boldsymbol{e}_i$, $\boldsymbol{e}_{pos(i)}$ to denote the current position and the position of historical token we want to copy respectively. Then, we construct $\boldsymbol{W_b} = [\boldsymbol{O}_{N\times d}, \boldsymbol{I}_N, \boldsymbol{O}_N] \in \mathbb{R}^{N\times(d+2N)}$ so that $\tilde{\boldsymbol{b}}_i = \boldsymbol{W_b}\tilde{\boldsymbol{x}}_i = \boldsymbol{e}_i$. Then, at the i-th step, the information newly recorded in the state space will be $\boldsymbol{v}_i \tilde{\boldsymbol{b}}_i^T = \boldsymbol{v}_i \boldsymbol{e}_i^T \in \mathbb{R}^{d\times N}$ and the updated state space will be $\boldsymbol{H}_i = \boldsymbol{H}_{i-1} + \sum_{j=1}^{i-1} \boldsymbol{v}_j \tilde{\boldsymbol{b}}_j^T + \boldsymbol{v}_i \tilde{\boldsymbol{b}}_i^T = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_i, \boldsymbol{O}_{d\times(N-i)}]$ thus at the last step, we can record all historical information in the state space by $\boldsymbol{H}_T = \sum_{j=1}^{N} \boldsymbol{v}_j \tilde{\boldsymbol{b}}_j^T = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_N]$. Then, at the output process, we can construct $\boldsymbol{W_c} = [\boldsymbol{O}_{N\times d}, \boldsymbol{O}_N, \boldsymbol{I}_N] \in \mathbb{R}^{N\times(d+2N)}$ so that $\tilde{\boldsymbol{c}}_i = \boldsymbol{W_c}\tilde{\boldsymbol{x}}_i = \boldsymbol{e}_{pos(i)}$. Thus, the output will be $\boldsymbol{y}_i = \boldsymbol{H}_i \tilde{\boldsymbol{c}}_i = \sum_{j=1}^{i} \boldsymbol{v}_j \boldsymbol{b}_j^T \boldsymbol{e}_{pos(i)} = \boldsymbol{v}_{pos(i)}$.

At the same time, we note that in the above process, the vectors $\boldsymbol{e}_i, \boldsymbol{e}_{pos(i)}$ to denote position in $\tilde{\boldsymbol{x}}_i$ are sparse. In fact, we only need to use two indices $p_i = i$ and $p_{pos(i)} = pos(i)$ to store them thus the total size to store all indices is $O(N)$. Additionally, $\boldsymbol{W_b}, \boldsymbol{W_c}$ are also sparse so we require at most $O(N)$ space to store these two matrices. Therefore, the model size we need is $O(Nd)$ that scales linearly with the length N. Thus, we complete our proof. □

Remark 1. *It should be noted that we can also degenerate a linear-size Mamba block into the aforementioned SSM module by deactivating the gating branch to achieve the COPY operation. For example, we can set $\boldsymbol{W}_1 = \boldsymbol{W}_3 = \boldsymbol{I}, \boldsymbol{b}_1 = \boldsymbol{0}, \boldsymbol{W}_2 = \boldsymbol{O}$ and $\boldsymbol{b}_2 = k\boldsymbol{1}$ where the constant k satisfies $\sigma(k) = 1$. In addition, we note that $\boldsymbol{x}_i \in [-M, M]^d$ and $p_i, p_{pos(i)} \in [1, N]$ thus the largest value involved in the aforementioned process will not exceed NM^2 (the largest value in hidden states), which is upper bounded by $O(poly(M, N))$. All parameters being upper bounded by $O(poly(M, N))$ means that the problem can also be solved by the same Mamba block with $log(N)$ precision, which has been also adopted in previous works (Merrill and Sabharwal, 2023b; Feng et al., 2024; Wen et al., 2024).*

A.4 Examples for DP Problems

We consider the problem of finding the minimum edit distance, where we aim to transform string $s^{(1)}$ into string $s^{(2)}$ with lengths $n_1 = |s^{(1)}|$ and $n_2 = |s^{(2)}|$, respectively. The costs for insertion, deletion, and substitution are a, b, and c, respectively. Obviously, the input sequences for this problem are $\{s^{(1)}, s^{(2)}\}$ and the scale of the problem is $\boldsymbol{n} = [n_1, n_2]^T$. Further, the state space is $\mathcal{I}_{\boldsymbol{n}} = \{(i, j) | 1 \le i \le n_1, 1 \le j \le n_2\}$. Let $\mathrm{dp(j,k)}$ represent the cost of transforming the first j characters of s_1 into the first k characters of s_2. The transition function can then be expressed as:

$$\mathrm{dp}(j,k) = \begin{cases} ak & \text{if } j = 0 \\ bj & \text{if } k = 0 \\ \min\big(\mathrm{dp}(j,k-1) + a, \mathrm{dp}(j-1,k) + b, & \\ \qquad \mathrm{dp}(j-1,k-1) + c\mathbb{I}[s_j^{(1)} \neq s_k^{(2)}]\big) & \text{otherwise} \end{cases}$$

Finally, the aggregation function selects $\mathrm{dp}(\mathrm{n}_1, \mathrm{n}_2)$ as the final answer. In the example above, the size of the state space, all intermediate values, and the lengths of the input strings will all be upper-bounded by $\mathrm{poly}(\mathrm{n}_1, \mathrm{n}_2)$. Moreover, the operations required by the involved functions can be approximated with polynomial efficiency by a constant-size MLP, as shown in Lemmas in Appendix A.6. Therefore, such DP problems satisfy Assumption 3. It can be referenced from Section 4.1 of Feng et al. (2024) for more detailed examples for DP problems.

A.5 Explanation of m-locality:

The most common arithmetic tasks satisfy the m-locality assumption, which are also frequently encountered in current LLM applications. Consider an arithmetic expression of length n; the CoT reasoning required to compute it step by step may have a total output length of $T = O(n^2)$. However, the window size m required for computing each intermediate state does not exceed n. For example, consider the expression $7 * (7 + 5 + 10) =$. The input length is $n = 10$, and its complete CoT output would be: $7 * (7 + 5 + 10) = 7 * (1 + 10) = 7 * 0 = 0$. As we can see, aside from the input part, the generation of each subsequent token requires a context window no larger than the input length n. Therefore, this task satisfies the m-locality.

Another example that satisfies the m-locality is the Edit Distance problem. When computing the 2D DP matrix, the number of entries (or tokens) required for each step does not exceed $m \leq \max\{n_1, n_2\}$. For instance: x g v | x g o <p> 0 2 4 , 2 0 2 , 4 2 3 , <p> 3 where <p> denotes the separate token to enclose the DP matrix arranged row by row. If we copy the input sequence when computing each row of the DP matrix, such as:

x g v | x g o <p> x g v | x g o 0 2 4 , x g v | x g o <p>
2 0 2 , x g v | x g o <p> 4 2 3 , <p> 3

then the entire process becomes strictly m-locality, where $m \leq 2n + 2$ with $n = |s_1| + |s_2|$ representing the input size. Meanwhile, the total CoT output length is roughly $O(n^2)$.

However, for the LIS problem, computing each state might require revisiting the entire input string, and the corresponding CoT output length itself is $T = O(n)$. This does not satisfy m-locality because m is at least n and cannot be significantly smaller than the output length T in terms of order.

A.6 Proof of Theorem 3

In this part, we first present the necessary lemmas before completing the proof of Theorem 4. In fact, these lemmas are very similar to those presented by Feng et al. (2024) in Appendix C.1 regarding MLP. The main difference is that we need to degenerate Mamba blocks to MLP and consider different activation functions (SiLU for Mamba instead of GELU). Thus, we only provide detailed proofs of these relevant lemmas when necessary.

Lemma 1 (Perform multiplication)**.** *For any $\epsilon > 0$ and $M > 0$, there exists Mamba block parameters with l_∞ norm upper bounded by $O(poly(M, 1/\epsilon))$ such that $|f(a, b) - ab| \leq \epsilon$ holds for all $a, b \in [-M, M]$.*

Proof. We first show that a two-layer MLP using the SiLU activation function can achieve the above operation. We use the same construction as in Lemma C.1. in Feng et al. (2024), except that we use the SiLU activation function instead of GELU. Specifically, let $g : \mathbb{R}^2 \to \mathbb{R}$ be a two-layer MLP with SiLU activation, and the hidden dimension is 4, then we can construct f as

$$g(a, b) = \frac{\lambda^2}{2}\left(\sigma\left(\frac{a+b}{\lambda}\right) + \sigma\left(\frac{-a-b}{\lambda}\right) - \sigma\left(\frac{a-b}{\lambda}\right) - \sigma\left(\frac{-a+b}{\lambda}\right)\right), \tag{36}$$

where λ is a scaling factor. In addition, considering $\sigma(x) = \frac{x}{1+e^{-x}}$, $\sigma'(x) = \frac{1+(x+1)e^{-x}}{(1+e^{-x})^2}$, $\sigma''(x) = \frac{e^{-x}(2+2e^{-x}+xe^{-x}-x)}{(1+e^{-x})^3}$, we have $\sigma(0) = 0$, $\sigma'(0) = \frac{1}{2}$, $\sigma''(0) = \frac{1}{2}$. Then, using the Taylor expansion with the Lagrange remainder, we can obtain that

$$\begin{aligned}
&\sigma\left(\frac{a+b}{\lambda}\right) + \sigma\left(\frac{-a-b}{\lambda}\right) - \sigma\left(\frac{a-b}{\lambda}\right) - \sigma\left(\frac{-a+b}{\lambda}\right) \\
&= \frac{1}{2!}\frac{1}{2}\left(\left(\frac{a+b}{\lambda}\right)^2 + \left(\frac{-a-b}{\lambda}\right)^2 - \left(\frac{a-b}{\lambda}\right)^2 - \left(\frac{-a+b}{\lambda}\right)^2\right) + R_2 = \frac{2ab}{\lambda^2} + R_2,
\end{aligned}$$

where R_2 is the second-order remainder term. Assuming that $\lambda > 2M$, we have $|\frac{\pm a \pm b}{\lambda}| < \frac{2M}{\lambda} < 1$ and

then

$$
\begin{aligned}
|R_2| &\leq \frac{4}{3!}\left(\frac{2M}{\lambda}\right)^2 \max_{x\in[-1,1]} |\sigma'''(x)| \\
&= \frac{4}{3!}\left(\frac{2M}{\lambda}\right)^2 \max_{x\in[-1,1]} \left|\frac{(x-3)e^{-x} - 4xe^{-2} + (x+3)e^{-3x}}{(1+e^{-x})^4}\right| \\
&\leq \frac{4}{3!}\left(\frac{2M}{\lambda}\right)^2 \frac{4e + 4e^2 + 4e^3}{(1+e^{-1})^4} \\
&\leq \frac{4}{3!}\frac{8M^3}{\lambda^3}\frac{81}{2} \\
&= \frac{216M^3}{\lambda^3}.
\end{aligned}
$$

Thus if we set $\lambda \geq \frac{216M^3}{2\epsilon}$ we will have $|g(a,b) - ab| \leq \frac{\lambda^2}{2}|R_2| \leq \epsilon$.

Then, we note that a Mamba block $\boldsymbol{f}$ defined as Eq (3) can degenerate into the above MLP g by deactivating its SSM branch. Specifically, we only need to set $\boldsymbol{W}_1$ to be zeros and $\boldsymbol{b} = \mathbf{1}$ so that the input of the SSM branch is a constant 1, that is,

$$
\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{W}_3 \cdot \mathrm{SSM}(\mathbf{1}) \odot \sigma(\boldsymbol{W}_2\boldsymbol{x} + \boldsymbol{b}_2).
$$

In the SSM module, we can set $\boldsymbol{W_b}$ to be zeros, that is, no new information will be retained in the hidden state. Following this, we set $\boldsymbol{d} = \mathbf{1}$ resulting that given $\boldsymbol{x} = \mathbf{1}$, we have $\mathrm{SSM}(\mathbf{1}) = \boldsymbol{y} = \boldsymbol{c}^T\boldsymbol{H} + \boldsymbol{d} \odot \boldsymbol{x} = \boldsymbol{c}^T\mathbf{1} + \mathbf{1} \odot \mathbf{1} = \mathbf{1}$. Thus the SSM branch can be deactivated and the Mamba block will degenerate into a two layer MLPs, that is,

$$
\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{W}_3\sigma(\boldsymbol{W}_2\boldsymbol{x} + \boldsymbol{b}_2). \tag{37}
$$

Furthermore, given $\boldsymbol{x} = [a, b]$, we can set $\boldsymbol{W}_2 \in \mathbb{R}^{4\times 2}$ and $\boldsymbol{W}_3 \in \mathbb{R}^{4\times 1}$ to meet the two-layer MLP g as Eq (36). Additionally, we note that all parameters of this Mamba block can be upper bounded by $O(poly(M, 1/\epsilon))$ under the l_∞ norm. Thus, we complete our proof. □

Remark 2. *It should be noted that here we have only provided one possible construction and this is not unique. For example, in the process of deactivating the SSM branch, we could also choose to make* $\boldsymbol{\Delta}$ *sufficiently large and correspondingly* $\tilde{\boldsymbol{A}}$ *sufficently small with* $\boldsymbol{A} \leq \mathbf{0}$ *so that the hidden states approximates zeros. In fact, the expressive power of an Mamba block with two branches should be stronger than that of a two-layer MLP since it already encompasses the latter. Nevertheless, we still provide one possible construction here.*

Lemma 2 (Approximate two-layer MLPs with ReLU)**.** *Let* $\boldsymbol{g} : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ *be a two-layer MLP with ReLU activation, and all parameters are upper bounded by* M*. Then, for any* $\epsilon > 0$ *, there exists a Mamba block* $\boldsymbol{f}$ *and parameters upper bounded by* $O(poly(M, 1/\epsilon))$ *in the* l_∞ *norm, such that for all* $\boldsymbol{x} \in \mathbb{R}^{d_1}$*, we have* $\|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{g}(\boldsymbol{x})\|_\infty \leq \epsilon$*.*

Proof. Similar to Lemma 1, once again, we deactivate the SSM branch, causing a Mamba block to degenerate into the form of Eq 37. Considering a two-layer MLP with a ReLU activation function denoted as $g(\boldsymbol{x}) = \overline{\boldsymbol{W}}_3\mathrm{ReLU}(\overline{\boldsymbol{W}}_2\boldsymbol{x})$ where $\overline{\boldsymbol{W}}_2 \in \mathbb{R}^{d\times d_1}$ and $\overline{\boldsymbol{W}}_3 \in \mathbb{R}^{d_2\times d}$, we can set similar parameters for the degenerated Mamba blcok, that is, we consider $\boldsymbol{W}_2 = \lambda\overline{\boldsymbol{W}}_2$, $\boldsymbol{W}_3 = \frac{1}{\lambda}\overline{\boldsymbol{W}}_3$ in Eq (37) where λ is some large constant. In order to prove the lemma, we need to show that $\|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{g}(\boldsymbol{x})\|_\infty \leq \epsilon$ with some λ upper bounded by $O(poly(M, 1/\epsilon))$.

Considering a scalar $z \in \mathbb{R}$, we firstly consider the upper bound of the following equation:

$$
\left|\mathrm{ReLU}(z) - \frac{1}{\lambda}\mathrm{SiLU}(\lambda z)\right| = \left|\max(z, 0) - \frac{z}{1 + e^{-\lambda z}}\right| = \frac{|z|}{e^{\lambda|z|} + 1} \leq \frac{1}{\lambda},
$$

where we use the fact that $e^x + 1 > x$ for any $x \geq 0$. Then, let $\boldsymbol{z} = \overline{\boldsymbol{W}}_2\boldsymbol{x}$, we can show that for any $\boldsymbol{z} \in \mathbb{R}^d$,

$$\left\|\overline{\boldsymbol{W}}_3\mathrm{ReLU}(\boldsymbol{z}) - \frac{1}{\lambda}\overline{\boldsymbol{W}}_3\mathrm{SiLU}(\lambda\boldsymbol{z})\right\|_\infty \leq \|\overline{\boldsymbol{W}}_3\|_\infty \left\|\mathrm{ReLU}(\boldsymbol{z}) - \frac{1}{\lambda}\mathrm{SiLU}(\lambda\boldsymbol{z})\right\|_\infty \tag{38}$$

$$\leq Md\left\|\mathrm{ReLU}(\boldsymbol{z}) - \frac{1}{\lambda}\mathrm{SiLU}(\lambda\boldsymbol{z})\right\|_\infty \tag{39}$$

$$\leq Md\max_{z\in\mathbb{R}}\left|\mathrm{ReLU}(z) - \frac{1}{\lambda}\mathrm{SiLU}(\lambda z)\right| \tag{40}$$

$$\leq \frac{Md}{\lambda}. \tag{41}$$

Then, if we set $\lambda > \frac{Md}{\epsilon}$, we will have $\|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{g}(\boldsymbol{x})\|_\infty \leq \epsilon$ and all parameters of the Mamba block is upper bounded by $O(poly(M, 1/\epsilon))$. Thus, we complete our proof. □

Remark 3. *We have proven that a Mamba block can approximate a two-layer MLP with ReLU activation function, and since the latter can perform many basic operations, including linear transformations and selection operations as constructed in Lemma C.3 and Lemma C.5 in Feng et al. (2024), we can use Lemma 2 to adopt the same construction, enabling the Mamba block to perform these operations. We present the following colloary more specifically, and the detailed proof can be found in the above mentioned part in Feng et al. (2024).*

Lemma 3 (Perform linear transformation, easily derived from Lemma 2 and Lemma C.3 in Feng et al. (2024))**.** *Let $\boldsymbol{W} \in \mathbb{R}^{d_2\times d_1}$ be any matrix used for implementing linear transformations upper bounded by M and $\boldsymbol{f} : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ be a Mamba block. Then, for any $\epsilon > 0$, there exist Mamba block parameters with l_∞ norm bounded by $O(poly(M, 1/\epsilon))$, such that for any $\boldsymbol{x} \in \mathbb{R}^{d_1}$, we have $\|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{W}\boldsymbol{x}\|_\infty \leq \epsilon$.*

Lemma 4 (Perform select operation, easily derived from Lemma 2 and Lemma C.4 in Feng et al. (2024))**.** *Define the selection function $\boldsymbol{g} : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ as follows:*

$$g(\boldsymbol{x}, \boldsymbol{y}, t) = \begin{cases} \boldsymbol{x} & if\ t > 0 \\ \boldsymbol{y} & if\ t < 0 \end{cases} \tag{42}$$

Let $\boldsymbol{f} : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ be a Mamba block. Then, for any $\epsilon > 0$, $\alpha > 0$, and $M > 0$, there exist Mamba parameters with l_∞ norm bounded by $O(poly(M, 1/\alpha, 1/\epsilon))$, such that for all $\boldsymbol{x} \in [-M, M]^d$, $\boldsymbol{y} \in [-M, M]^d$, and $t \in [-\infty, -\alpha] \cup [\alpha, +\infty]$, we have $\|\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}, t) - g(\boldsymbol{x}, \boldsymbol{y}, t)\|_\infty \leq \epsilon$.

Next, we show that one Mamba layer or several Mamba layers can implement indicator functions through the select operation. We mainly focus on the usual indicator functions $\mathbb{I}[a \neq b]$, $\mathbb{I}[a > b]$ and $\mathbb{I}[a < b]$.

Lemma 5 (Perform indicator function)**.** *Define the indicator function $\mathbb{I}(a, b, \circ) : \mathbb{R}^2 \times \{\neq, >, <\} \to \{0, 1\}$ where $a, b \in [-M, M]$. The output of the function will be 1 if $a \circ b$ is satisfied otherwise the output will be 0. Let $f : \mathbb{R}^2 \to \mathbb{R}$ be a Mamba block. Then, for any $\epsilon > 0$, there exist Mamba parameters with l_∞ norm upper bounded by $O(poly(M, 1/\epsilon))$, such that for any $a, b \in [-M, M]$ and $\circ \in \{\neq, >, <\}$, we have $\|f(a, b) - \mathbb{I}(a, b, \circ)\|_\infty \leq \epsilon$.*

Proof. We first show that a Mamba block can implement $\mathbb{I}[a > b]$ and $\mathbb{I}[a < b]$. For $\mathbb{I}[a > b]$, it is equivalent to consider $g(1, 0, a - b)$ where $g(\cdot)$ defined in Lemma 4. So firstly we can use a linear layer with appropriate parameters $\boldsymbol{W}_0, \boldsymbol{b}_0$ to convert the input $[a, b]$ into the vector $[1, 0, a - b]$. Then we can use Lemma 4 to implement $\mathbb{I}[a > b]$ by changing the parameters of the first linear layer from $\{\boldsymbol{W}_1, \boldsymbol{b}_1\}$ to $\{\boldsymbol{W}_1\boldsymbol{W}_0, \boldsymbol{b}_1 + \boldsymbol{W}_1\boldsymbol{b}_0\}$. The proof for $\mathbb{I}[a < b]$ is similar as well.

Noticing that $\mathbb{I}[a \neq b] = 1 - (1 - \mathbb{I}[a > b]) \cdot (1 - \mathbb{I}[a < b])$, we can implement $\mathbb{I}[a \neq b]$ through the following layers: Firstly, we can use one Mamba block to implement $1 - \mathbb{I}[a > b]$ and $1 - \mathbb{I}[a < b]$ simultaneously, where the hidden dimension will be 8 and the output is a vector $[1-\mathbb{I}[a > b], 1-\mathbb{I}[a < b]]$. Then, another Mamba block is constructed to implement the multiplication $(1 - \mathbb{I}[a > b]) \cdot (1 - \mathbb{I}[a < b])$

according to Lemma 1 and the appropriate outermost linear layer parameters are chosen to simultaneously achieve multiplication by a negative sign and addition of a bias of 1, where the hidden dimension will be 4 and the output will be $\mathbb{I}[a \neq b]$. Thus, we complete our proof. □

Furthermore, following the setting of Feng et al. (2024), we also make the following assumption:

Assumption 3. *Given input sequences $s^{(1)}, s^{(2)}, \ldots, s^{(N)}$, we consider the following constraints for the DP problem:*

- *For any $i \in \mathcal{I}_{\boldsymbol{n}}$, there exists constants $N_{\boldsymbol{s}}$, N_{dp} and $N_{\mathcal{A}}$ such that $|\mathcal{I}_i| \leq N_{\boldsymbol{s}}$, $|\mathcal{V}_{\mathrm{dp}(i)}| \leq N_{\mathrm{dp}}$ and $|\mathcal{A}_{\boldsymbol{n}}| \leq N_{\mathcal{A}}$.*
- *The size of the state space $|\mathcal{I}_{\boldsymbol{n}}|$, the embeddings of all tokens in the input sequence, all intermediate DP values $(\mathrm{dp}(i)$ for $i \in \mathcal{I}_{\boldsymbol{n}})$, and the final answer A can all be polynomially upper bounded by the problem size $\boldsymbol{n}$.*
- *The functions used to solve the DP problem, including the function $f_{\mathcal{I}}$ to determine the next state, the transition function $f_{\mathcal{T}}$, the aggregation function $f_{\mathcal{A}}$ and $\mathcal{A}(\boldsymbol{n})$ can all be approximated with polynomial efficiency by a constant-size MLP (with the SiLU activation function).*

Remark 4. *The first constraint of the Assumption 3 illustrates that the number of input tokens and previous DP values used in the transition function at each step can be upper bounded by $N_{\boldsymbol{s}}$ and N_{dp}. In addition, the number of DP values used in aggregation is at most $N_{\mathcal{A}}$. This is reasonable because the number of inputs for solving each state in a DP problem should be finite. The second constraint is a restriction on the magnitude of the intermediate values, allows that all involved inputs and outputs used in functions can be represented by the log-precision model. The third constraint allows a constant-sized degenerated Mamba to implement functions required to solve the DP, which has been proved by above Lemmas 1-5. In fact, due to the first constraint, the sizes of inputs and outputs of these functions will be a constant related to $\{N_{\boldsymbol{s}}, N_{\mathrm{dp}}, N_{\mathcal{A}}\}$.*

In fact, Assumption 2 covers many dynamic programming (DP) problems commonly encountered in real-world scenarios, such as basic arithmetic operations, the Longest Increasing Subsequence (LIS), and Edit Distance (ED). The sizes of these tasks grow polynomially with respect to the input size. While the functions involved in solving subproblems of these tasks may be non-smooth, we have shown in Lemmas 1-5 that they can all be approximated by MLPs with polynomial efficiency. Therefore, Assumption 2 is reasonable for DP problems.

Now, based on the basic operations that can be implemented by the Mamba blocks as discussed above, we present the proof of Theorem 4:

Theorem 9 (Perform DP problems with CoT)**.** *Considering any DP problem and given input sequences that satisfies Assumption 3, for any integer $T \in \mathbb{N}$, there exists several Mamba layers with size $O(T)$, such that the answer generated by the Mamba layers will be correct when the length of the answer is no more than T.*

Proof. Firstly, we illustrate the input format for the DP problem. We follow the embedding format in the proof of Theorem 4.7 in Feng et al. (2024), that is, assuming that the input at any step of solving the DP problem using CoT is a sequence of tokens embedded as follows:

$$\boldsymbol{x}_t^{(0)} = \left[\boldsymbol{e}_t^{\mathrm{input}}, \boldsymbol{e}_t^{\mathrm{state}}, \boldsymbol{e}_t^{\mathrm{dp}}, \boldsymbol{e}_t^{\mathrm{answer}}, \boldsymbol{e}_t^{\mathrm{sep}}, t, 1\right],$$

where the specific value of each part is depend on the content represented by the current token. More specifically, each part can be described as:

- If the current position denotes a input token, then we set e_t^{input} as the embedding of the input and simultaneously set $\boldsymbol{e}_t^{\mathrm{state}} = \boldsymbol{e}_t^{\mathrm{dp}} = \boldsymbol{e}_t^{\mathrm{answer}} = \boldsymbol{e}_t^{\mathrm{sep}} = \boldsymbol{0}$.
- If the current position is the final answer, then $\boldsymbol{e}_t^{\mathrm{anwser}}$ denotes the embedding of the answer and we set $\boldsymbol{e}_t^{\mathrm{input}} = \boldsymbol{e}_t^{\mathrm{state}} = \boldsymbol{e}_t^{\mathrm{dp}} = \boldsymbol{e}_t^{\mathrm{sep}} = \boldsymbol{0}$.

- If the current position denotes the j-th separator | between input sequences , then we set $\boldsymbol{e}_t^{\text{sep}} = \boldsymbol{e}_j$ and $\boldsymbol{e}_t^{\text{input}} = \boldsymbol{e}_t^{\text{state}} = \boldsymbol{e}_t^{\text{dp}} = \boldsymbol{e}_t^{\text{answer}} = \mathbf{0}$.
- If the current position denotes an intermediate DP state, then we use $\boldsymbol{e}_t^{\text{state}}$ to denote the embedding of the DP state and $\boldsymbol{e}_t^{\text{dp}}$ denotes the corresponding value. Similarly, other part will be set to be $\mathbf{0}$.
- The scalar t denotes the current position in the whole sequence, which holds the value for all above cases.

We illustrate that here we use a concatenation operation to replace the residual connection defined in Eq (2), which is a technique also used by Feng et al. (2024); Yang et al. (2024) in similar proofs. This is because, from the perspective of expressive capability, the two operations are equivalent: the output of a Mamba block $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ concatenated with the input (that is, $[\boldsymbol{y}, \boldsymbol{x}]$) can also be represented using the residual connection: $\boldsymbol{g}([\boldsymbol{x}, \mathbf{0}]) + [\mathbf{0}, \boldsymbol{x}] = [\boldsymbol{y}, \mathbf{0}] + [\mathbf{0}, \boldsymbol{x}] = [\boldsymbol{y}, \boldsymbol{x}]$ where $\boldsymbol{g} : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ is another Mamba block and part of its parameters to be same as $\boldsymbol{f}$ and others are set to be $\mathbf{0}$. Conversely, the concatenation can implement residual connection by using a linear projection.

Here, we show our construction of several Mamba layers to solve the DP problem, which is composed of different blocks to perform different tasks:

Block 1: The first block aims to calculate the problem size $\boldsymbol{n}$ and the embedding of the next state $\boldsymbol{e}_t^{\text{next\_state}}$. This process can be described as follows:

- **Compute the problem size n:** (i) First, we can replicate the position of the token $t_{sep,1}, t_{sep,2}, \ldots, t_{sep,N}$ using the COPY operation. This can be achieved with a Mamba layer of size $O(Ntd)$ according to Theorem 3; (ii) Then, we calculate the size of the problem as $\boldsymbol{n} = [t_{\text{sep,1}} - 1, t_{\text{sep,2}} - t_{\text{sep,1}} - 1, \ldots, t_{\text{sep,N}} - t_{\text{sep,N}-1} - 1]$, which can be done by applying a linear transformation using one Mamba layer, as shown by Lemma 3.
- **Obtain the next state $\boldsymbol{e}^{\text{next\_state}}$:** According to Assumption 3, the function $\boldsymbol{e}^{\text{next\_state}} = f(\boldsymbol{n}, \boldsymbol{e}^{\text{state}})$ which determines the next state, can be approximated by constant-sized MLPs. Thus, this can also be implemented by having several Mamba layers degenerate into MLPs.

The output after this step can be written as:

$$\boldsymbol{x}_t^{(1)} = [\boldsymbol{e}_t^{\text{input}}, \boldsymbol{e}_t^{\text{state}}, \boldsymbol{e}^{\text{next\_state}}, \boldsymbol{e}_t^{\text{dp}}, \boldsymbol{e}_t^{\text{answer}}, \boldsymbol{e}_t^{\text{sep}}, \boldsymbol{n}, t, 1]$$

Block 2: The second block is mainly constructed to find the indices of input tokens and intermediate DP values that are needed to calculate the DP value corresponding to $\boldsymbol{e}^{\text{next\_state}}$. Specifically, this can be described as follows:

- **Calculate the needed indices:** We calculate the positions of the input token $\boldsymbol{p}_t^{\boldsymbol{s}} = I_{\boldsymbol{s}}(\boldsymbol{n}, \boldsymbol{e}^{\text{state}})$ and the positions of tokens that correspond to needed DP values $\boldsymbol{p}_t^{\text{dp}} = I_{\text{dp}}(\boldsymbol{n}, \boldsymbol{e}^{\text{state}})$. If $I_{\boldsymbol{s}}(\boldsymbol{n}, \boldsymbol{e}^{\text{state}}) = \emptyset$ or $I_{\text{dp}}(\boldsymbol{n}, \boldsymbol{e}^{\text{state}}) = \emptyset$, we set the positions to be a special value γ. According to Assumption 3, these two functions can be done by constant-size MLPs thus can be approximated by degenerated Mamba layers.
- **Set the flag:** (i) Set the flag f_t^{answer} based on whether the DP value of the current state is needed in the final aggregation function. This can be achieved by several Mamba layers with Assumption 3 that the function $\mathcal{A} = f(\boldsymbol{n}, \boldsymbol{s})$ can be approximated by MLPs and additionally using Lemma 5 to implement $\mathcal{I}[\boldsymbol{e}_t^{\text{state}} \neq \boldsymbol{e}_j^{\text{state}}]$ where $\boldsymbol{e}_j^{\text{state}} \in \mathcal{A}$. (ii) Set the flag f_t^{state} to denote whether the current state is the last state. This can be implemented by checking $\mathbb{I}[\boldsymbol{e}_t^{\text{next\_state}} \neq \mathbf{0}]$ with Mamba layers using Lemma 5.

The output result after this step can be written as:

$$\boldsymbol{x}_t^{(2)} = [\boldsymbol{e}_t^{\text{input}}, \boldsymbol{e}_t^{\text{state}}, \boldsymbol{e}^{\text{next\_state}}, \boldsymbol{e}_t^{\text{dp}}, \boldsymbol{e}_t^{\text{answer}}, \boldsymbol{e}_t^{\text{sep}}, \boldsymbol{n}, \boldsymbol{p}_t^{\boldsymbol{s}}, \boldsymbol{p}_t^{\text{dp}}, f_t^{\text{answer}}, f_t^{\text{state}}, t, 1]$$

Block 3: This block is designed to calculate the DP value for the next state. In detail, the implementation involves the following steps:

- **Check the flag:** We check the flag f_t^{state} using several Mamba layers using 5 to implement $\mathbb{I}[f_t^{\text{state}} \neq 1]$ using Lemma 5. If $f_t^{\text{state}} = 1$, the current denote is the last state and we just need to set $\boldsymbol{p}_t = \gamma\mathbf{1}$ where $\boldsymbol{p}_t$ denotes $\boldsymbol{p}_t^{\boldsymbol{s}}$ and $\boldsymbol{p}_t^{\text{dp}}$, which implies $I_{\boldsymbol{s}}(\boldsymbol{n}, \boldsymbol{e}^{\text{state}}) = \emptyset$ and $I_{\text{dp}}(\boldsymbol{n}, \boldsymbol{e}^{\text{state}}) = \emptyset$, that is, no input tokens or DP values are needed.

- **Obtain the needed embeddings:** If $f_t^{\text{state}} \neq 1$, then (i) We copy the input token embeddings $\boldsymbol{e}^{\text{input}}$ at positions $\boldsymbol{p}_t^{\boldsymbol{s}}$. This COPY operation can be implemented by a Mamba layer of size $O(N_s td)$ using Theorem 3; (ii) Simultaneously, we copy the embeddings of DP values at positions $\boldsymbol{p}_t^{\text{dp}}$, which can be achieved by a Mamba layer of size $O(N_{\text{dp}} td)$. If the position is empty, we just need to check $\mathbb{I}[\boldsymbol{p}_t \neq \gamma\mathbf{1}]$ and set the needed embeddings $\boldsymbol{e}^{\text{input}}$ or $\boldsymbol{e}^{\text{dp}}$ to be some special token. Totally, the size of Mamba layers in this step is $O((N_s + N_{\text{dp}})td)$.

- **Calculate the DP value:** We calculate the DP value $\boldsymbol{e}_t^{\text{next\_state}}$ for the next state with the Assumption 3 that the transition function can be approximated by several Mamba layers using Lemma 2.

The output result after this step can be written as:

$$\boldsymbol{x}_t^{(2)} = [\boldsymbol{e}_t^{\text{input}}, \boldsymbol{e}^{\text{next\_state}}, \boldsymbol{e}_t^{\text{next\_dp}}, \boldsymbol{e}_t^{\text{answer}}, \boldsymbol{e}_t^{\text{sep}}, \boldsymbol{n}, f_t^{\text{answer}}, f_t^{\text{state}}, t, 1]$$

Block 4: The last block is constructed to implement the final aggregation function and output the final answer. Specifically, the steps are as follows:

- **Check the flag:** We identify whether the current state is the last state by checking $\mathbb{I}[f_t^{\text{state}} \neq 1]$ by using Lemma 5. If $f_t^{\text{state}} = 1$, then all intermediate DP values have been solved and we need to compute the final answer.

- **Obtain the needed embeddings:** We collect the DP value embeddings $\boldsymbol{e}^{\text{dp}}$ of these tokens whose $f^{\text{answer}} = 1$, which can be achieved by COPY operation according to Theorem 3 with one Mamba layer of size $O(N_{\mathcal{A}} td)$.

- **Generate the final answer:** Finally, we compute the answer by implementing the aggregation function, which can be achieved by constant-size MLPs according to Assumption 3, thus can also be achieved by several degenerated Mamba layers.

In summary, given a sequence length t and equipped with CoT, the parameter size required by the Mamba layers to generate the correct answer at each step is $O(\tilde{N}td)$, where $\tilde{N} = \max\{N, N_{\boldsymbol{s}} + N_{\text{dp}}, N_{\mathcal{A}}\}$ is a constant independent of t, that is, the size of the Mamba layer scales linearly with t. Thus, we complete our proof. □

A.7 Proof of Theorem 5

Theorem 10 (Perform m-locality DP problems with CoT)**.** *Consider any m-locality DP problem and given input sequences that satisfies Assumption 3, for any integer $T \in \mathbb{N}$, there exists several Mamba layers with size $O(m)$, such that the answer generated by the Mamba layers will be correct when the length of the answer is no more than T.*

Proof. The overall proof construction approach is similar to that of Theorem 4, with the only difference being that under the assumption of m-locality, when performing the COPY operation, the constructed Mamba only needs to focus on at most m tokens preceding the current position. This results in the size of the Mamba layers only needing to be $O(\tilde{N}md)$. □

A.8 More Details of Experiments

For the copy task experiments, we mainly refer to the setup by Jelassi et al. (2024). For Transformers, we select the GPT-NeoX architecture (Andonian et al., 2023) while for Mamba we use the Mamba GitHub repository(Gu and Dao, 2023). More specifically, for the left part of Figure 3, we configure 10 layers for the Transformer (TF-126M) and 5 layers for TF-63M, with both having a hidden size of 1024 and RoPE (Su et al., 2024) as the positional encoding. For Mamba models, we configured 20 layers and a hidden size of 1024 for Mamba-135M, 20 layers and a hidden size of 720 for Mamba-D-67M, 10 layers and a hidden size of 1024 for Mamba-L-67M, and 40 layers and a hidden size of 512 for Mamba-LD-69M. We use an online sampling batch size of 8 and set the maximum context length to 220, meaning each example often contains multiple instances. AdamW(Loshchilov, 2017) is chosen as the optimizer with a learning rate of 1e-5 and weight decay of 0.1. We set $N_{\min} = 10$ and $N_{\max} = 30$ for all models.

For the right part of Figure 3, the Transformer and Mamba setups match the aforementioned configurations for TF and Mamba. Moreover, we set $[N_{\min}, N_{\max}]$ to $[5, 10]$, $[10, 20]$, $[20, 30]$ and $[30, 40]$ for sequence lengths of 10, 20, 30 and 40 respectively.

For the phonebook tasks, we mainly follow the setting of Jelassi et al. (2024). We use pretrained Mamba models of size 370M, 1.4B and 2.8B(Gu and Dao, 2023) and for the Transformer baseline, we use the pretrained 410M Pythia model(Biderman et al., 2023). These models have been pre-trained on the Pile(Gao et al., 2020) and use the same tokenizer. The Mamba models generally have slightly lower perplexity on the training set for a given size (Jelassi et al., 2024). For Mamba-2.8B(pre), the prompt at the begining is just like:

" The following is a phonebook with the form: Gary Battle: 8444797678 Gary Gallegos: 9960330831. Remeber the phone number of Joseph Perry. Here is the phonebook:..."

For the CoT task experiments, we mainly follow the setup of Feng et al. (2024); Yang et al. (2024). For the LIS task, we investigate different task lengths $L = \{10, 30, 50, 70$ which denotes the length of the input sequence to solve. While for the Arithmetic task, we select the task length as $L = \{4, 5, 6, 7\}$. Here, the task length refers to the number of steps required to incrementally compute the arithmetic expression. An example when $L = 4$ is: $4 \times (8 - 6/3) = 4 \times (8 - 2) = 4 \times 6 = 24$. The model dimensions is selected from $d = \{32, 64, 128, 256\}$ and the number of layers is set to 3 by default for Transformers. While for Mamba, under each setting, we adjust the number of layers to match or slightly exceed the size of the Transformer within the same d, with roughly two Mamba layers corresponding to one Transformer layer. All models are trained for 300 epochs using AdamW(Loshchilov, 2017) with a learning rate of 1e-4 and weight decay of 0.01. For the results shown in the Figures 6, 8 and 10, we report the average test accuracy over the last five epochs as the final accuracy. We run all experiments three times and reported the average results. Furthermore, our experiments were conducted on four 24GB NVIDIA GeForce RTX 3090 GPUs and were completed within five days. More results are shown in Appendix A.9.

A.9 More experiment results

For a more comprehensive presentation and easier comparison, more results on the LIS task are provided in Figures 7 and 8.

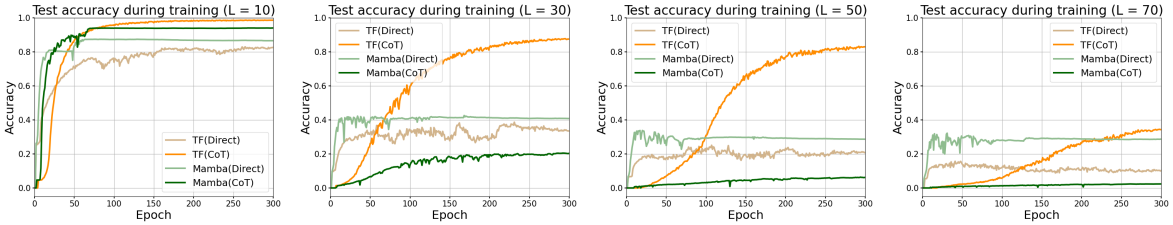

Figure 7: Test accuracy on LIS tasks during training when the task length $L = 10, 30, 50, 70$ and $d = 256$ (TF denotes Transformer)

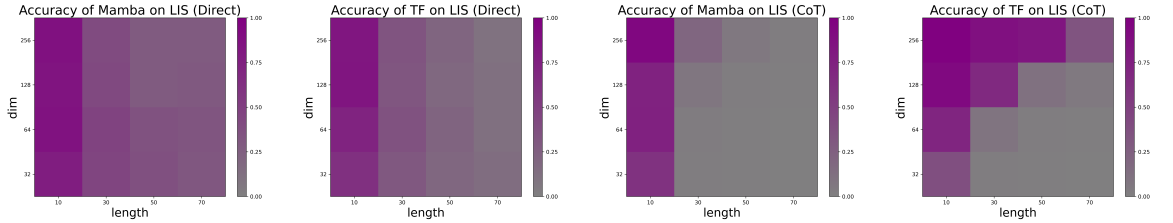

Figure 8: Test accuracy on LIS tasks of Mamba and Transformer across different task lengths and model sizes (with/without CoT).

Similarly, additional results on arithmetic tasks are presented in Figures 9 and 10. We find that for arithmetic tasks, given the same limited amount of training data as in the LIS task (training set size of 51,200), neither Transformer nor Mamba can effectively learn the task under direct setting (close to random guessing). However, under the CoT training setting, both models are able to learn the task to some extent. Nevertheless, Mamba's performance still lags behind that of the Transformer.

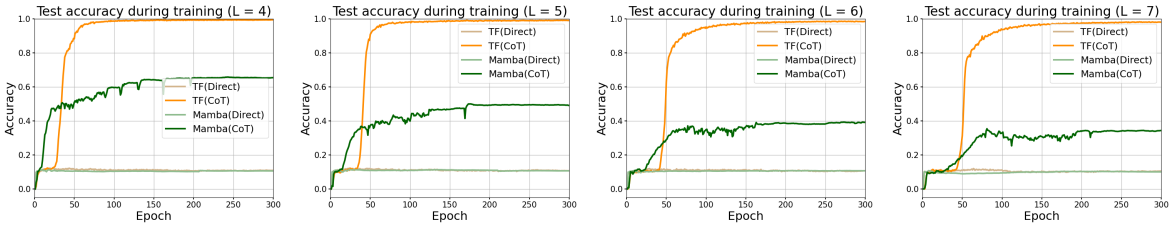

Figure 9: Test accuracy on Arithmetic tasks during training when the task length $L = 4, 5, 6, 7$ and $d = 256$ (TF denotes Transformer). Without CoT, the model's accuracy is close to that of random guessing.

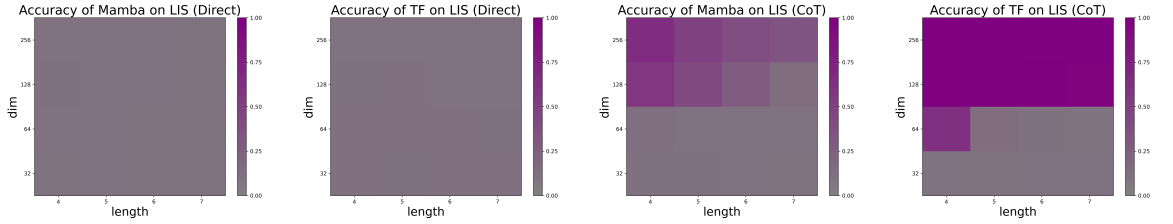

Figure 10: Test accuracy on Arithmetic tasks of Mamba and Transformer across different task lengths and model sizes (with/without CoT).)

More experiments on Hybrid models. One may ask whether we can improve Mamba to bridge its gap with Transformers. As illustrated before, from the theoretical perspective, the expressive capacity

of Mamba is constrained by its limited horizon and its inability to dynamically scale its reasoning cost (capacity) with the problem size. One intuition to close the gap is to combine Mamba with Transformer layers to form a hybrid architecture, which is also adopted in practice (Waleffe et al., 2024; Wen et al., 2024; Lieber et al., 2024). To illustrate this intuition, we further conduct experiments on LIS tasks. Specifically, while keeping the overall parameter count roughly unchanged, we replace the first and last layers of Mamba with Transformer layers, which is denoted as "Hybrid". Additionally, we also consider a variant where only the first layer is replaced by a Transformer layer, denoted as "Hybrid-". The results are shown in Figure 11 and 12.

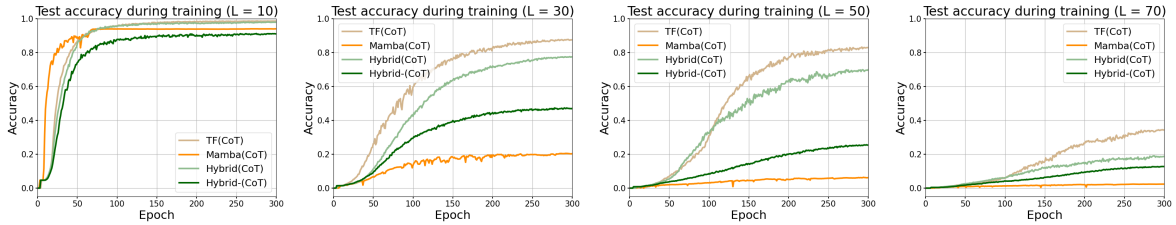

Figure 11: Test accuracy on LIS tasks during CoT training when the task length $L = 10, 30, 50, 70$ and $d = 256$ (TF denotes Transformer).

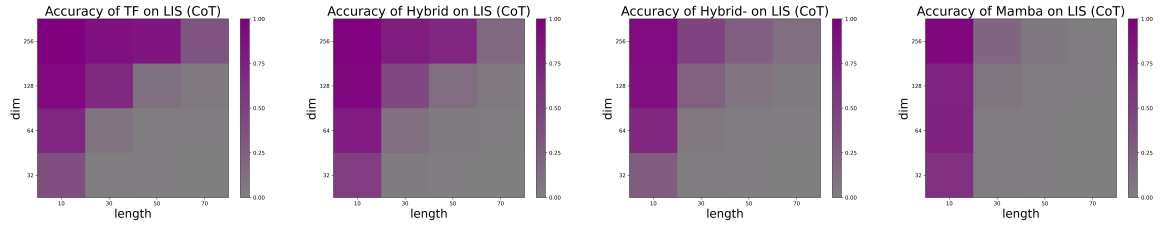

Figure 12: Test accuracy on LIS tasks of different models across different task lengths and model sizes (with CoT).

Overall, the hybrid architecture achieves a trade-off between the performance of the Transformer and Mamba architectures. This suggests that in scenarios where the problem size or complexity is not very large, appropriately incorporating some Transformer layers into Mamba can effectively improve its performance and leverage the strengths of both architectures.