Revealing and Mitigating the Local Pattern Shortcuts of Mamba

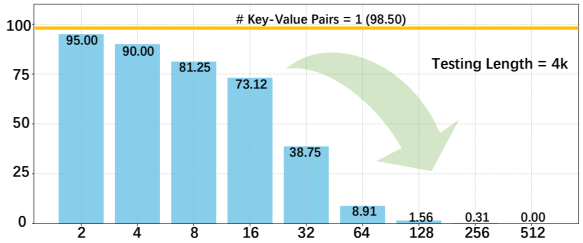
Wangjie You[1*], **Zecheng Tang**[1*], **Juntao Li**[1†], **Lili Yao**[2], **Min Zhang**[1]
[1]School of Computer Science and Technology, Soochow University
[2]Machine learning platform department, Tencent
{wjyouuu,zctang}@stu.suda.edu.cn
{ljt,minzhang}@suda.edu.cn; liliyao@tencent.com

Abstract

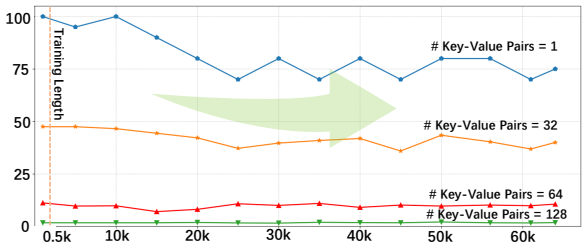
Large language models (LLMs) have advanced significantly due to the attention mechanism, but their quadratic complexity and linear memory demands limit their performance on long-context tasks. Recently, researchers introduced Mamba, an advanced model built upon State Space Models (SSMs) that offers linear complexity and constant memory. Although Mamba is reported to match or surpass the performance of attention-based models, our analysis reveals a performance gap: Mamba excels in tasks that involve localized key information but faces challenges with tasks that require handling distributed key information. Our controlled experiments suggest that the inconsistency arises from Mamba's reliance on **local pattern shortcuts** across model scales (10M to 1.4B), which enable Mamba to remember local key information within its limited memory but hinder its ability to retain more dispersed information. Therefore, we introduce a global gate module into the Mamba model to address this issue. Experiments on extensive synthetic tasks, as well as real-world tasks, demonstrate the effectiveness of our method. Notably, with the introduction of only **4M** extra parameters, our approach enables the Mamba model (130M) to achieve a significant improvement on tasks with distributed information, increasing its performance **from below 5% to 80%**.

1 Introduction

In recent years, State Space Model (SSM) has emerged as a promising successor to the attention-based models (Vaswani et al., 2017) for long sequence modeling due to its linear computational complexity and constant memory requirements (Gu et al., 2022a; Gupta et al., 2022; Gu et al., 2022b; Smith et al., 2023; Dao and Gu, 2024). Different from the attention mechanism, which stores information for each token and performs pairwise computations between them, SSMs use a fixed-size state space to store history. This allows all computations to involve only the constant-sized state space. Mamba (Gu and Dao, 2023), built upon SSMs, is claimed to have achieved performance on par with, or even surpassing, that of attention-based models with the same parameters across language modeling and various synthetic tasks (Dao and Gu, 2024; Waleffe et al., 2024; Chen et al., 2024).



(a) Performance as key-value pairs numbers increase.



(b) Performance as testing length increases.

Figure 1: Mamba exhibits two distinct trends under different settings. The y-axis represents accuracy, while the x-axis in Fig.(a) shows the number of key-value pairs in the context with a testing length of 4K. In Fig.(b), the x-axis represents the testing length.

However, we observe an intriguing discrepancy in Mamba's performance on two settings of the MQAR[1] task: one requires the model to retrieve information from a local segment (single Key-Value pair) within the context, while the other requires retrieving dispersed information (multiple Key-Value pairs) from the context. As shown in Fig 1(a),

[*] Equal Contribution.
[†] Corresponding author.

[1]A synthetic task for testing a model's retrieval capability.

Findings of the Association for Computational Linguistics: ACL 2025, pages 12156–12178

Mamba can effectively retrieve information from the local segment (single Key-Value pair within a 4K context), even with a context length of up to 60K. However, in tasks that require extracting dispersed (a few Key-Value pairs within a 4K context) or locally dense information (a large number of Key-Value pairs within a 4K context), Mamba's performance is significantly affected and deteriorates sharply as the information density and dispersion increase. Additionally, as shown in Fig 1(b), compared to the effects of information density and dispersion within the context, Mamba is relatively less affected by the context length.

In this paper, we conduct a controlled study to better understand the characteristics of Mamba under different context settings. We start by analyzing Mamba's performance on different synthetic tasks, covering different information densities and dispersions. Our findings reveal that Mamba relies on **local pattern shortcuts** to extract the desired information from the context, which manifests in two aspects: (1) **positional shortcuts**, i.e., Mamba tends to extract information from specific positions; and (2) **n-gram shortcuts**, i.e., Mamba tends to utilize specific high-frequent training templates to locate information. These shortcuts make the model less robust to perturbed inputs, thereby further limiting Mamba's ability to generalize to complex tasks. As a result, as previous works revealed (Amos et al., 2023; Park et al., 2024; Ben-Kish et al., 2024; Arora et al., 2024c), Mamba performs well on tasks it trained on but struggles significantly on unseen tasks. Furthermore, we explain potential reasons for these shortcuts from two perspectives: the limited recurrent state size of the Mamba model and the constrained selectivity mechanism of SSMs.

To mitigate the aforementioned issues, we propose a global gate mechanism for the Mamba model. Specifically, we design an input-dependent global gating module for Mamba and observe significant improvements in its performance on complex synthetic tasks and the language modeling task. Notably, by introducing **just 4M additional parameters** to the 130M-sized Mamba model, Mamba can achieve a breakthrough **from below 5% to 80%** on high information density synthetic tasks, significantly narrowing the performance gap between Mamba and attention-based models.

2 Background

2.1 State Space Model (SSM)

Structured state space sequence models (S4) (Gu et al., 2021, 2022b; Goel et al., 2022; Ma et al., 2023; Hasani et al., 2023; Smith et al., 2023), represent a recent class of sequence models closely related to classical state space models. These models are inspired by a specific continuous system that facilitates the mapping of a one-dimensional function or sequence $x(t) \in \mathbb{R}$ to an output $y(t) \in \mathbb{R}$ through an implicit latent state $h(t) \in \mathbb{R}^N$. Concretely, S4 models are characterized by four parameters (Δ, A, B, C), which define a sequence-to-sequence transformation as follows:

$$\begin{aligned} h'(t) &= \overline{A}h(t-1) + \overline{B}x(t) \\ y(t) &= Ch(t) \end{aligned} \tag{1}$$

where (Δ, A, B) are the discrete parameters, $A \in \mathbb{R}^{N\times N}, B \in \mathbb{R}^{N\times 1}, C \in \mathbb{R}^{1\times N}$, $\overline{A} = f_A(\Delta, A)$, $\overline{B} = f_B(\Delta, A, B)$. Additionally, $f_A(\cdot), f_B(\cdot)$ are the pre-defined discretization functions.

2.2 Selective State Space Model (Mamba)

Selective State Space Model, as known as Mamba, is different from previous SSMs where the model's dynamics remain constant over time, it can efficiently update its hidden state based on the current input by introducing selective parameters. Specifically, Mamba accomplishes this by employing specialized trainable linear layers that map the input to the matrices $\overline{B}$, $\overline{C}$, and the time step Δt for each processing step. Mamba conditions the discrete time-variant matrices dynamically based on the input as follows:

$$\Delta_t = \tau(S_\Delta X_t), \quad \overline{B}_t = S_B X_t, \quad \overline{C}_t = (S_C X_t)^T, \tag{2}$$

$$\overline{A}_t = \exp(A_t \Delta_t), \quad \overline{B}_t = B_t \Delta \tag{3}$$

where Δt represents the discretization step, τ denotes the softplus function, and S_Δ, S_B, and S_C are linear transformation functions. This enhancement empowers Mamba to execute more flexible sequence modeling, particularly for tasks demanding extensive historical information e.g., the Selective Copying task (Arjovsky et al., 2016) and the Induction Heads task (Olsson et al., 2022), surpassing the performance of other SSMs. Further discussions on other SSM variants and efficient model structures can be found in Appendix A.2.

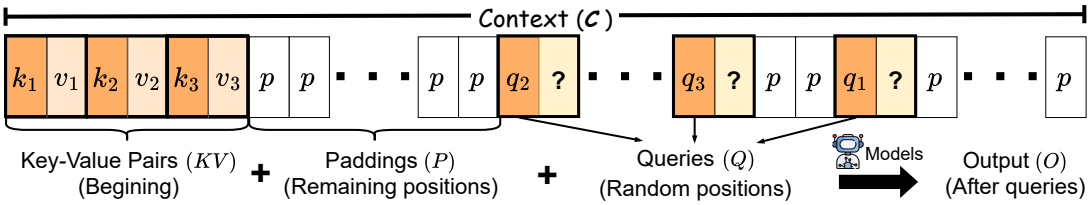


Figure 2: Illustration of MQAR Task.

2.3 Multi-Query Associative Recall

To make evaluations more controllable and eliminate the influence of the models' intrinsic knowledge, synthetic tasks are often employed (Hsieh et al., 2024). We conduct a further discussion of the synthetic tasks in Appendix A.1. Among them, the Multi-Query Associative Recall (MQAR) is a widely adopted synthetic task for SSMs. In MQAR, an input x is structured as a sequence of bigrams representing key-value pairs, which are randomly drawn from a predefined dictionary. Queries, i.e., the keys of key-value pairs, are then appended to this sequence, requiring the model to retrieve the corresponding value for the queried key. As depicted in Fig. 2, formally, the input context $\mathcal{C} = (c_0, \ldots, c_{N-1})$ consists of N tokens, where $c_i \in \mathcal{V}$ and $\mathcal{V}$ is the vocabulary of the model. We define N as the context length, representing the length of the input sequence. The input sequence $\mathcal{C}$ can be divided into three parts: key-value pairs KV, queries $\mathcal{Q}$, and padding tokens $\mathcal{P}$. The key-value pairs are $KV = \{(k_1, v_1), (k_2, v_2), \ldots, (k_n, v_n)\}$, where n is the predefined number of key-value pairs. In the standard MQAR task, these key-value pairs are placed at the beginning of the sequence. Queries are represented as $\mathcal{Q} = \{q_1, q_2, \ldots, q_n\}$, where $q_i = k_i$, and are inserted at random positions after the key-value pairs. Padding tokens are defined as $\mathcal{P} = \mathcal{C} \setminus (\mathcal{Q} \cup KV)$, and they occupy the remaining positions in $\mathcal{C}$, filled with random tokens. The objective of the MQAR task is to predict:

$$\mathcal{O}_i = \arg\max_{o_i \in \mathcal{V}} f_\theta(o_i \mid q_i, KV, \mathcal{P}),$$

where the model aims to output the most probable token $\mathbf{o_i}$ from the vocabulary, given the padding tokens, key-value pairs, and the query $\mathbf{q_i}$. MQAR requires models to memorize key-value pairs in their hidden state, which presents a significant challenge for *rnn-based* models, as they maintain a fixed-size state to handle all historical information.

3 Analysis of Local Pattern Shortcuts in Mamba

Previous studies (Gu and Dao, 2023; Arora et al., 2024a,b) have shown that Mamba's success stems from its data-dependent features, where Mamba can dynamically gate the previous information based on the current input. However, based on our preliminary study (as shown in Fig. 1), we observe that Mamba performs poorly when the key information with the context becomes denser or more dispersed, regardless of various context lengths. To discover the underlying reasons, we study the changes in the state space of the Mamba model during the inference process. In Sec. 3.1, we first reformulate Mamba's process of assigning weights to each token into an attention-like matrix. Then, we test the mamba model with the synthetic retrieval tasks and analyze the model's state space during the inference process. Specifically, we utilize the 130M version of Mamba in all the experiments and design three different testing sets based on the MQAR task: (1) Positional Pattern Change, which alters the distribution of information in the context, moving beyond the previous MQAR task that places key information at the beginning (Sec. 3.3); (2) N-gram Gathering, which controls the degree of aggregation of key information in the context by introducing more information within local segments, rather than solely testing the model's recall ability of single token (Sec. 3.2); and (3) Noise Injection, which adds noise tokens into the key information to perturb the model predictions, aiming to test the robustness of Mamba model (Sec. 3.4).

3.1 Reformulating Selection Process of Mamba into Attention-like Matrix

Ali et al. suggests that Mamba's selection process can be reformulated into an attention-like matrix. Specifically, give the sequence $Y = \{y_1, y_2, \cdots, y_L\}$ that contains L tokens, we leverage Eq. 1 to calculate each y_i and reformulate the calculation process into matrix multiplication,

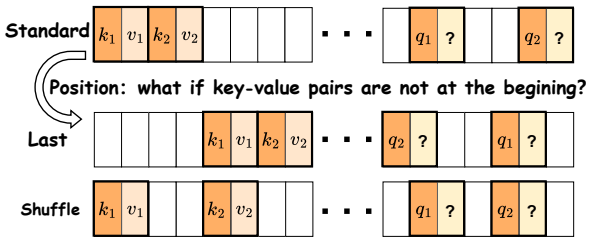


Figure 3: MQAR task with different positional patterns.

which can be written as:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_L \end{bmatrix} = \begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,L} \\ \alpha_{2,1} & \alpha_{2,2} & \cdots & \alpha_{2,L} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{L,1} & \alpha_{L,2} & \cdots & \alpha_{L,L} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_L \end{bmatrix}, \quad (4)$$

where $\alpha_{i,j} = C_i \left(\prod_{k=j+1}^{i} \bar{A}_k \right) \bar{B}_j \ (i, j \in [1, L])$.

Then, we can transform Eq. 4 into an attention-like matrix by substituting and expanding $\alpha_{i,j}$:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_L \end{bmatrix} = \underbrace{\begin{pmatrix} C_1\bar{B}_1 & 0 & \cdots & 0 \\ C_2\bar{A}_2\bar{B}_1 & C_2\bar{B}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ C_L \prod_{k=2}^{L} \bar{A}_k \bar{B}_1 & C_L \prod_{k=3}^{L} \bar{A}_k \bar{B}_2 & \cdots & C_L\bar{B}_L \end{pmatrix}}_{} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_L \end{bmatrix}, \quad (5)$$

where the portion enclosed by the **underbrace** represents the weights allocated by Mamba across the sequence, and we visualize this part to explore Mamba's state space.

3.2 Positional Pattern Change

Task Description As depicted in Fig. 3, in the standard MQAR task, all Key-Value pairs are placed at the beginning of the sequence. This setup may lead the model to learn that it only requires "remembering" content from the initial portion of the sequence during training. To avoid such a fixed pattern of information distribution, we concentrate them at the end of the sequence (Last) as well as disperse the key-value pairs from the beginning to arbitrary positions throughout the sequence (Shuffle). As depicted in Fig. 3, after adopting the aforementioned settings, i.e., Last and Shuffle, queries ($\mathcal{Q}$) are inserted at the random positions in the remaining padding sequence ($\mathcal{P}$). Then, we train the Mamba model **separately** with training data containing three different positional patterns and evaluate it on **all** three test sets.

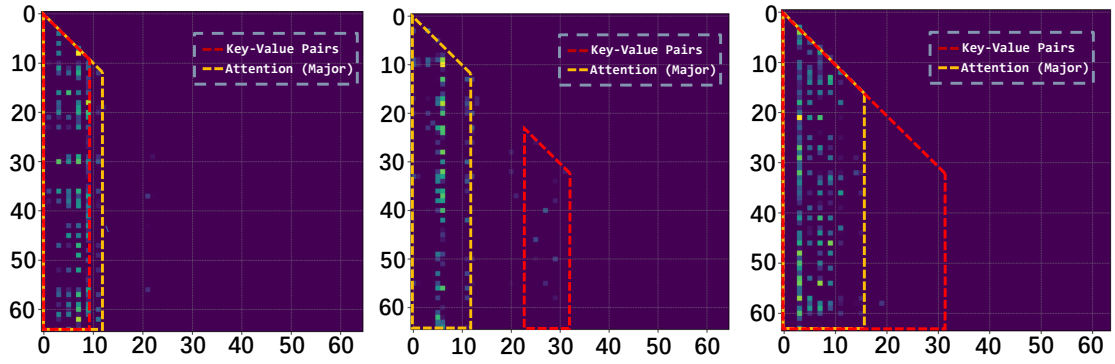
Results As shown in Table 1, we can observe that when trained on the standard MQAR setup, Mamba achieved near-perfect accuracy on the in-domain test set (standard), achieving 99.72 points. In the other test settings, i.e., Last and Shuffle, Mamba only managed 15.44% and 22.37% accuracy, respectively. However, when experimenting with the Mamba model on other settings, e.g., training with Shuffle data and testing with all three testing sets, Mamba performs consistently across all test settings, achieving or exceeding 50% accuracy. This finding reveals that **Mamba tends to leverage information position patterns from the training data, allowing it to excel in tasks where the training and testing phases follow the same fixed template.** The visualized attention matrix of Mamba trained on the standard and test on all three position patterns, as depicted in Fig. 4, further supports this conclusion. The attention matrix represents the weights allocated by Mamba across the sequence, despite the relocation of the key-value pairs, Mamba consistently attends to the beginning of the sequence, a behavior aligned with its training pattern but misaligned with the actual key-value pairs' position pattern. This results in near-perfect performance in the in-domain i.e., standard setting, but a noticeable decline in out-of-domain patterns.

Test \ Train	Standard	Last	Shuffle
Standard	99.72	82.64	90.80
Last	15.44	99.35	78.09
Shuffle	22.37	54.08	80.98

Table 1: Model performance on the Positional Pattern Change setting, where we report the prediction accuracy.

3.3 N-gram Gathering

Task Description In the standard MQAR task, models are required to predict the correct value given a key, where both the key and value are single tokens, i.e., $(k_i, v_i) \in KV$ and $|k_i| = |v_i| = 1$. We refer to this configuration as the *K1V1* setting, primarily evaluating the model's 2-gram recall capability. However, addressing only 2-gram recall is insufficient, as the amount of information to be recalled is minimal, and learning specific 2-gram pairings is relatively easy. Moreover, most real-world entities involve multiple tokens, and testing 2-gram capability alone fails to reflect performance on other tasks. Therefore, to increase the amount of information to be recalled, we propose the N-gram Gathering setting. As shown in Fig. 5, we increase the number of tokens required for recall in both the Key and Value portions, i.e.,



(a) Standard-Standard (b) Standard-Last (c) Standard-Shuffle

Figure 4: The attention-like matrices of Mamba-130M that are trained on standard MQAR task and are tested on all three testing sets, i.e., Standard, Last, and Shuffle. We plot the results of the 22nd layer of the model. Lighter colors indicate higher attention scores at specific positions. The red dashed line represents the location of the key-value pairs, while the yellow dashed line indicates where the model attends to the most.

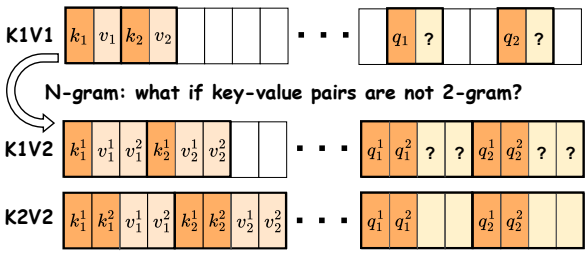


Figure 5: MQAR task with different n-gram patterns.

$|k_i| = N, |v_i| = M$ $(N > 1, M > 1)$ We refer to such a configuration as the *KNVM* setting. Specifically, we set up three data configurations: $K1V1$, $K1V2$, $K2V2$. We train the model on each of these settings **separately** and then evaluate it across **all** three testing sets.

Results As shown in Table 2, Mamba exhibits strong performance when the number of key-value tokens in the training set matches or exceeds those in the test set, e.g., training with $K1V1$ or $K1V2$ and testing with $K1V1$. However, its performance deteriorates significantly when the number of value tokens in the test set surpasses those encountered during training, e.g., training with $K1V1$ and testing with $K1V2$ or $K2V2$. This indicates that Mamba tends to learn simple patterns in tasks, e.g., learning to respond based on the single anchor token in N key tokens, which hinders its generalization ability on other complex tasks. Furthermore, Mamba's success would not be due to true n-gram recall but rather an **over-reliance on the structural cues provided by the special characters or templates**, which can also explain its failure in out-of-domain n-gram setting. We refer to this phenomenon as the n-gram shortcut of Mamba.

Test \ Train	$K1V1$	$K1V2$	$K2V2$
$K1V1$	99.95	99.81	66.82
$K1V2$	0.00	99.96	96.66
$K2V2$	0.00	99.90	99.98

Table 2: Model performance on the N-gram Gathering setting, where we report the prediction accuracy.

3.4 Noise Injection

Task Description We further explore Mamba's robustness, specifically its ability to generalize beyond the shortcuts mentioned above. As shown in Fig. 6, we place n sets of Key-Value pairs at the beginning of the sequence and divide these Key-Value pairs into four regions: $KV = \{(k_1, v_1), (k_2, v_2), (k_3, v_3), (k_4, v_4)\}$. Then, two settings are adopted: $K1V1^*$ and $K2V2^*$. In $K1V1^*$, all the keys k_i are identical, i.e., $k_1 = k_2 = k_3 = k_4$, and we utilize symbol $\overline{k_i}$ to denote those tokens. In $K2V2^*$, the last token of k_i are identical. All the values v_i are different in the above two settings. Then, we let the model predict the corresponding value by providing k_i.

Results As shown in Fig. 6(a), in terms of the $K1V1^*$ setting, we observe that the model's performance during testing closely aligns with the training patterns. Specifically, when all tokens are placed at the beginning of the sequence during training (Standard MQAR), the model tends to retrieve information from the front, primarily focusing on the first 25% of the sequence. Conversely, if all key information is positioned toward the end of the sequence during training (Last MQAR), the

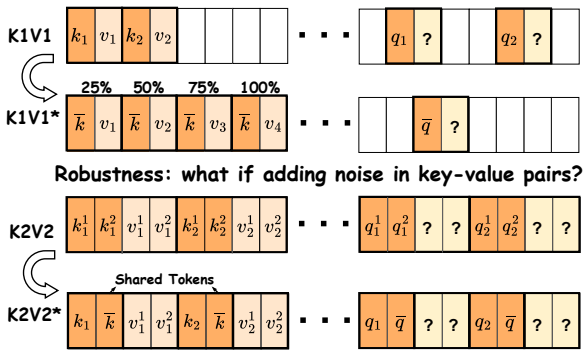


Figure 6: MQAR task with the injection of noise. * denotes the robustness setting.

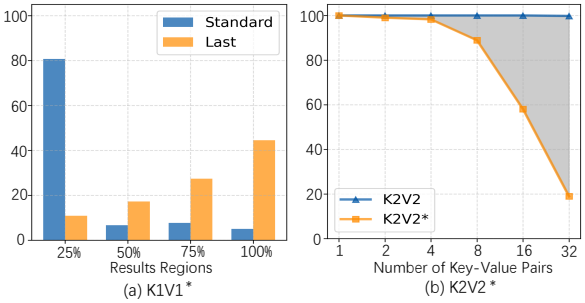


Figure 7: (a) Results under the $K1V1^*$ setting after training with different position modes (Standard MQAR and Last MQAR), where the x-axis represents the region ratios. (b) Results under the $K2V2^*$ setting, with the x-axis indicating the number of key-value pairs for models trained on standard $K2V2$ data.

model tends to retrieve information from the latter parts of the sequence, concentrating mainly on the 75% to 100% regions. This behavior highlights Mamba's **over-fitting to the positional patterns learned during training, which may lead to a reliance on positional shortcuts rather than recall capability.** Besides, as shown in Fig. 6(b), we can observe that as the number of Key-Value pairs increases, Mamba maintains an accuracy exceeding 90% in the presence of noise when there are four Key-Value pairs. However, as the amount of noisy Key-Value pairs increases, Mamba's performance declines sharply. This indicates that Mamba relies on the partial high-frequent information within the keys for its predictions. Therefore, when noise overwhelms this critical information, the model's performance declines dramatically.

4 Mitigating the Shortcuts of Mamba

4.1 Key to Selectivity of Mamba

For *attention-based* models, the recurrent state grows with the length of the sequence, enabling perfect recall accuracy but at the cost of efficiency. In contrast, *RNN-based* models maintain a fixed recurrent state size, which makes it critical to optimize the use of their limited memory resources. Mamba distinguishes itself by efficiently balancing the memory-recall trade-off through its data-dependent design. However, our previous experiments revealed that this selective mechanism can inadvertently introduce shortcuts. So the question is: **how can we mitigate the shortcut phenomenon while preserving the advantage of Mamba's fixed state-space size?** To better understand this, let us revisit Mamba's state-space update equation:

$$h_t = \bar{A}h_{t-1} + \bar{B}x_t = e^{A\Delta t}h_{t-1} + \Delta t B x_t, \quad (6)$$

where $\Delta_t = \tau(S_\Delta X_t)$. Here, Δ_t is central to Mamba's selectivity as it simultaneously governs the behavior of the matrices $\bar{A}$ and $\bar{B}$ through a linear transformation, ultimately determining how the recurrent state is updated. In the original Mamba, Δ_t is generated from the current input via a short convolution function followed by two linear transformations. The primary goal of this process is to extract local features and establish contextual relationships among tokens before entering the state space module. However, as the short convolution only captures relationships within a limited scope, its reliance can lead to shortcuts in scenarios where important information is distributed across distant positions. In such cases, the model may overly depend on simple features learned during training (e.g., positional cues) rather than capturing deeper, more meaningful dependencies.

4.2 Incorporating Global Gate Mechanism

To address the shortcut issue, we propose incorporating additional global information into Δ_t through a fine-grained global gate mechanism. Specifically, we introduce a long convolution module to capture distant contextual information and integrate its output with the original Δ_t. This enhanced process is formulated as follows:

$$\Delta_t = \tau\Big(\big(\mathbf{W}_2 \cdot \sigma\big(\mathbf{W}_1 \cdot \mathrm{Conv}_{\mathrm{short}}(\mathbf{X}_t)\big)\big) \odot \sigma\big(\mathrm{Conv}_{\mathrm{long}}(\mathbf{X}_t)\big)\Big), \quad (7)$$

where $\mathrm{Conv}_{\mathrm{short}}$ and $\mathrm{Conv}_{\mathrm{long}}$ denote the short and long convolution operations, respectively. The function τ represents the softplus activation function, $\mathbf{W}_1$ and $\mathbf{W}_2$ are linear transformation matrices, σ is the nonlinear activation function (specifically SiLU (Ramachandran et al., 2017)), and $\odot$

Models	Scale	Shuffle	Std-Last	Std-Shuffle	K2V2	K2V2-Robustness	K4V8-Shuffle
Pythia (Biderman et al., 2023)	*133m*	99.82	93.75	94.31	99.99	99.99	99.99
Hyena (Poli et al., 2023)	*153m*	✗	✗	✗	77.62	65.92	22.51
RWKV (Peng et al., 2023)	*153m*	✗	✗	✗	85.99	72.62	6.57
Mamba (Gu and Dao, 2023)	*129m*	80.98	15.44	22.37	99.98	66.01	✗
w/ *mimetic_init* (Trockman et al., 2024)	*129m*	82.57	16.11	20.42	99.96	80.37	✗
w/ 2×*State Size*	*130m*	88.57	40.22	31.88	99.84	78.90	✗
w/ 4×*State Size*	*134m*	96.92	35.89	32.88	99.84	57.11	✗
w/ *Global Gate*	*133m*	90.45	41.97	35.73	99.06	81.46	80.54

Table 3: Performance of models on variations of MQAR tasks. ✗ denotes that the model fails with this setting with an accuracy lower than 5%. ✗ indicates that the model fails in this setting, with an accuracy lower than 5%. *State Size* refers to the Mamba model with an increased state space size, which was originally set to 16.

Models	Scale	Std-Std	Std-Last	Std-Shuffle
Mamba	*13.8m*	98.94	17.48	20.81
w/ *GA*	*14.3m*	99.33	23.21	36.33
Mamba	*370m*	99.94	18.66	23.62
w/ *GA*	*384m*	100.0	30.10	47.52
Mamba	*1.4b*	99.64	21.21	26.73
w/ *GA*	*1.4b*	99.61	32.16	48.84

Table 4: The results of Mamba model at three different model scales i.e.,13m, 370m, and 1.4b on position change tasks, *GA* denotes the global gate strategy.

denotes element-wise multiplication.

The output of the long convolution serves as a gating mechanism that applies a global gate to the original Δ_t. By incorporating this global information, Mamba's selectivity becomes better aligned with global decision-making rather than being constrained to local patterns. This refinement enables the model to mitigate shortcut dependencies and capture more meaningful long-range relationships.

5 Experiment

5.1 Experimental Settings

Datasets We selected tasks where the original Mamba exhibited clear shortcuts and performed poorly. The settings are the same as outlined in Sec. 3. For the Shuffle and $K2V2$ settings, models are trained and tested on the same corresponding setting. We evaluate the model's in-domain capabilities in scenarios with increased information density and dispersion. In contrast, the Last and Std-Shuffle tasks test the model's out-of-domain performance, where the testing mode is inconsistent with the training one, i.e., training on the standard position pattern and testing on the Last and Shuffle pattern. In the $K2V2$-Robustness and $K4V8$-Shuffle settings, we evaluate whether the models can mitigate the influence of noise as well as handle tasks with both high information density and divergence. We further investigate the impact of the global gate strategy on downstream task performance, additional details can be found in Appendix B.

Baselines We adopt one representative *attention-based* model, Pythia (Biderman et al., 2023), along with two *RNN-based* models, Hyena (Poli et al., 2023) and RWKV (Peng et al., 2023), at a comparable scale. Our goal is to investigate whether models with different architectures exhibit similar local pattern shortcuts as Mamba. For Mamba models, Trockman et al. (2024) introduced a mimetic initialization technique to enhance Mamba's recall ability, which we adopt as a baseline. Additionally, increasing the state size intuitively allows the model to retain more historical information, reducing past data compression and thereby decreasing reliance on shortcuts. More details on model training hyperparameters can be found in Appendix H.

5.2 Main Result

Syntactic Tasks Table 3 presents the results on syntactic tasks, with particular focus on shortcut scenarios. The attention-based model, *Pythia*, achieves near-perfect performance across all tasks by effectively capturing long-range token-to-token dependencies. In contrast, RNN-based models such as Hyena and RWKV struggle on the MQAR task due to their fixed state sizes. Notably, neither Hyena nor RWKV exhibits the positional pattern shortcut observed in Mamba, as they do not display a significant gap between in-domain and out-of-domain performance. Moreover, although both models trail Mamba in standard $K2V2$ tasks, they perform comparably in the $K2V2$-Robustness setting, suggesting that Mamba is more susceptible to noise—likely due to its reliance on n-gram shortcuts. These findings imply that Mamba's shortcut behavior is closely tied to its unique selective mechanism, which prioritizes local patterns over

Scale	Models	Wiki. ppl ↓	LAMBDA acc ↑	PIQA acc ↑	Hella. acc ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc ↑	Openbook. acc ↑
130M Params ***3B tokens***	Pythia	40.94	22.96	61.21	27.90	51.14	43.60	18.09	14.20
	Mamba2	43.18	20.80	60.55	28.17	52.49	39.98	18.69	16.60
	Mamba	40.46	21.91	62.13	28.70	52.25	43.60	18.52	14.80
	w/ *GA*	38.92	22.80	62.62	28.79	52.41	42.38	18.69	16.20
370M Params ***15B tokens***	Pythia	24.25	35.36	65.78	32.41	51.70	51.47	19.88	18.80
	Mamba2	28.23	32.48	66.70	31.82	51.17	49.07	20.31	19.20
	Mamba	23.48	34.29	68.77	35.13	51.46	51.30	21.33	19.20
	w/ *GA*	22.43	35.07	67.90	35.16	51.14	53.24	22.70	20.80
790M Params ***30B tokens***	Pythia	23.12	40.46	69.32	34.68	53.35	56.34	23.78	18.60
	Mamba2	23.02	36.13	68.77	35.49	52.25	54.12	21.84	21.40
	Mamba	23.25	37.47	68.99	35.21	51.54	54.17	23.72	20.80
	w/ *GA*	21.33	39.84	69.64	36.55	52.49	56.73	24.40	19.00

Table 5: The results of models on downstream tasks, *GA* denotes the global gate strategy.

the global context because of the constrained Δ_t. Regarding Mamba models, while increasing the recurrent state size generally enhances performance, the gains are not consistently proportional, and the mimetic initialization method does not yield a significant boost. In contrast, integrating global information via the Global Gate consistently improves robustness without inflating the parameter count. The Global Gate strategy outperforms the original Mamba on all tasks and excels in out-of-domain scenarios, achieving 81.46% in $K2V2$-Robustness and 80.54% in $K4V8$-Shuffle, thereby effectively mitigating reliance on local shortcuts while handling high-information-density tasks.

Mamba Shortcuts across Model Scales We extend our analysis to include a broader range of model scales, beyond the 130M parameter configuration, to better understand how these shortcuts behave across different model capacities. Specifically, we select a small (10M parameters) and a large (1.4B parameters) version of Mamba and test them in the positional pattern change scenario, where we observe significant shortcut phenomena.

All models are trained on the standard position pattern and tested on three different position patterns. The results are shown in Table 4. It can be observed that the shortcut phenomenon occurs across Mamba models of different scales, and this phenomenon does not diminish as the model size increases. On the contrary, interestingly, we find that the impact of shortcuts may become more pronounced in larger models. For example, in the *Std-Last* task, the 13M Mamba outperforms the 370M Mamba. We attribute this to overparameterization, which aligns with the ideas presented by Chen et al.. They suggest that Mamba models may fail to generalize beyond the training length due to the excessive state capacity. Notably, global gate alleviates the shortcut phenomenon across all model scales, demonstrating the robustness of our method. We further investigate the performance of our proposed global gate strategy on downstream tasks in Appendix B and provide insight analysis of the trade-off between the local and global gate mechanisms on information processing in Appendix D.

Downstream Tasks As shown in Table 5, Mamba has already demonstrated notable success in downstream tasks, outperforming the *attention-based* baseline model, Pythia. The introduction of Global Gate further strengthens Mamba's capabilities in language modeling and commonsense reasoning. The Global Gate mechanism consistently delivers improvements across all evaluated tasks, highlighting its effectiveness in diverse domains. The most significant gains are observed in language modeling, where the perplexity is reduced by 1.54 compared to the original Mamba. This reduction underscores the model's enhanced ability to capture long-range dependencies while mitigating over-reliance on local patterns. Further exploration of downstream tasks, especially recall-intensive tasks, is presented in Appendix B, with a detailed case study provided in Appendix D.

6 Conclusion

In this work, we extend the MQAR task to investigate the underlying behavior of Mamba. Our controlled experiments reveal that Mamba relies on local pattern shortcuts at different model scales. To address this issue, we introduce a fine-grained selection mechanism in the Mamba model by incorporating global information into the decision-making factor Δ_t. Experiments on both existing and newly

proposed synthetic tasks show that our method effectively mitigates the shortcut phenomenon in Mamba across model scales ranging from 13.8M to 1.4B parameters. Further experiments on real-world tasks further demonstrate the effectiveness of our approach. Our findings suggest that for *RNN-based* models with a fixed recurrent state size, efficiently utilizing the available state space is far more critical than simply increasing capacity.

Limitation

Our approach is primarily focused on experimentally analyzing the shortcut phenomenon in Mamba; further exploration of theoretical insights is needed. The proposed method aims to address the shortcut issues observed in Mamba on synthetic tasks. However, further improving performance on downstream tasks may require additional adaptations and comprehensive testing across a wider range of models with varying scales. While preliminary results indicate that similar shortcuts were not present in other *rnn-based* models, further validation is needed to determine whether our findings generalize across diverse architectures.

Acknowledgements

We want to thank all the anonymous reviewers for their valuable comments. This work was supported by the National Science Foundation of China (NSFC No. 62206194), the Natural Science Foundation of Jiangsu Province, China (Grant No. BK20220488), and the Young Elite Scientists Sponsorship Program by CAST (2023QNRC001).

References

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

Ido Amos, Jonathan Berant, and Ankit Gupta. 2023. Never train from scratch: Fair comparison of long-sequence models requires data-driven priors. *arXiv preprint arXiv:2310.02980*.

Martin Arjovsky, Amar Shah, and Yoshua Bengio. 2016. Unitary evolution recurrent neural networks. In *The International Conference on Machine Learning (ICML)*, pages 1120–1128.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. 2024a. Zoology: Measuring and improving recall in efficient language models. In *The International Conference on Learning Representations (ICLR)*.

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. 2024b. Simple linear attention language models balance the recall-throughput tradeoff. In *The International Conference on Machine Learning (ICML)*.

Simran Arora, Aman Timalsina, Aaryan Singhal, Benjamin Spector, Sabri Eyuboglu, Xinyi Zhao, Ashish Rao, Atri Rudra, and Christopher Ré. 2024c. Just read twice: closing the recall gap for recurrent language models. *arXiv preprint arXiv:2407.05483*.

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. 2016. Using fast weights to attend to the recent past. *Advances in Neural Information Processing Systems (NeurIPS)*, 29.

Assaf Ben-Kish, Itamar Zimerman, Shady Abu-Hussein, Nadav Cohen, Amir Globerson, Lior Wolf, and Raja Giryes. 2024. Decimamba: Exploring the length extrapolation potential of mamba. *Preprint*, arXiv:2406.14528.

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

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on Artificial Intelligence*, volume 34.

Yingfa Chen, Xinrong Zhang, Shengding Hu, Xu Han, Zhiyuan Liu, and Maosong Sun. 2024. Stuffed mamba: State collapse and state capacity of rnn-based long-context modeling. *Preprint*, arXiv:2410.07145.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

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

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. 2021. A mathematical framework for Transformer circuits. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2021/framework/index.html.

Daniel Y Fu, Tri Dao, Khaled Kamal Saab, Armin W Thomas, Atri Rudra, and Christopher Re. 2023. Hungry hungry hippos: Towards language modeling with state space models. In *The Eleventh International Conference on Learning Representations*.

Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2021. A framework for few-shot language model evaluation.

Karan Goel, Albert Gu, Chris Donahue, and Christopher Ré. 2022. It's raw! audio generation with state-space models. In *The International Conference on Machine Learning (ICML)*.

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

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

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

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

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

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

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

Ramin Hasani, Mathias Lechner, Tsun-Hsuan Wang, Makram Chahine, Alexander Amini, and Daniela Rus. 2023. Liquid structural state-space models. In *The International Conference on Learning Representations (ICLR)*.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. 2024. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*.

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

Gregory Kamradt. 2023. Needle In A Haystack - pressure testing LLMs. *Github*.

Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. 2019. Openceres: When open information extraction meets the semi-structured web. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3047–3056.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Xuezhe Ma, Chunting Zhou, Xiang Kong, Junxian He, Liangke Gui, Graham Neubig, Jonathan May, and Luke Zettlemoyer. 2023. Mega: Moving average equipped gated attention. In *The International Conference on Learning Representations (ICLR)*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *Preprint*, arXiv:1609.07843.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*.

Amirkeivan Mohtashami and Martin Jaggi. 2023. Landmark attention: Random-access infinite context length for Transformers. In *Workshop on Efficient Systems for Foundation Models @ ICML*.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. 2022. In-context learning and induction heads. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1525–1534.

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 *The International Conference on Machine Learning (ICML)*.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, et al. 2023. RWKV: Reinventing RNNs for the Transformer era. *arXiv preprint arXiv:2305.13048*.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. 2023. Hyena hierarchy: Towards larger convolutional language models. *arXiv preprint arXiv:2302.10866*.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. 2017. Swish: A self-gated activation function. *arXiv preprint arXiv:1710.05941*, 7(1):5.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial Winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. 2023. Simplified state space layers for sequence modeling. In *The International Conference on Learning Representations (ICLR)*.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. 2023. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Asher Trockman, Hrayr Harutyunyan, J Zico Kolter, Sanjiv Kumar, and Srinadh Bhojanapalli. 2024. Mimetic initialization helps state space models learn to recall. *arXiv preprint arXiv:2410.11135*.

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

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

Shida Wang and Qianxiao Li. 2024. Stablessm: Alleviating the curse of memory in state-space models through stable reparameterization. *Preprint*, arXiv:2311.14495.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. 2024. Gated Linear Attention Transformers with hardware-efficient training. In *The International Conference on Machine Learning (ICML)*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Wei Zhang and Bowen Zhou. 2017. Learning to update auto-associative memory in recurrent neural networks for improving sequence memorization. *arXiv preprint arXiv:1709.06493*.

A Related Work

A.1 Synthetic Task

Synthetic tasks have played a crucial role in advancing language modeling, serving as controlled, simplified, and purposefully constructed benchmarks for assessing specific model capabilities. By providing a controlled environment, these tasks enable researchers to isolate particular challenges and evaluate how models handle them, offering deeper insights into their underlying mechanisms. Such tasks not only facilitate the identification of strengths and weaknesses in language models but also drive innovation and optimization in model architecture and training strategies. In contrast to more complex and realistic benchmarks, synthetic tasks offer greater flexibility in regulating various factors like sequence length, task complexity, and input structure. This flexibility makes them invaluable for probing specific behaviors of language models without interference from large-scale parametric knowledge or the unpredictability of real-world data. As a result, they serve as ideal testing grounds for understanding how models manage tasks that require attention over long sequences or the processing of intricate patterns.

Numerous synthetic tasks have been developed to test different dimensions of model performance. For example, copying and selective copying tasks (Jing et al., 2019) evaluate a model's memory retention and replication abilities, while tasks involving induction heads (Olsson et al., 2022) examine its capacity to infer relationships in context. Others, such as passkey retrieval (Mohtashami and Jaggi, 2023), needle-in-a-haystack (Kamradt, 2023), and associative recall (Graves et al., 2014; Ba et al., 2016), have been instrumental in testing how well large language models (LLMs) handle long-range dependencies, particularly in extremely long sequence contexts. In addition, Hsieh et al. propose a novel synthetic benchmark RULER to evaluate long-context language models.

The *associative recall* (AR) task, inspired by psychological models of how humans associate and retrieve information, has been a focal point of early neural network research aimed at developing systems capable of associative recall. With the advent of large language models, many researchers have argued that the ability of LLMs to perform *in-context* learning is, at least in part, attributable to the associative recall capabilities embedded in attention mechanisms (Elhage et al., 2021; Olsson et al., 2022). More recently, several notable recurrent neural network architectures have been evaluated using synthetic versions of the associative recall task (Graves et al., 2014; Ba et al., 2016; Zhang and Zhou, 2017, inter alia).

Our work builds upon and extends one of these tasks, specifically the Multi-Query Associative Recall (MQAR) (Arora et al., 2024a). In contrast to the standard associative recall task, MQAR is designed to more closely resemble the complexities of natural language processing. It introduces multiple key-value pairs and challenges models to retrieve the correct associations despite distractors, thereby offering a more rigorous test of a model's ability to manage selective attention and long-range dependencies within dynamic, varied input structures. This task pushes the boundaries of synthetic evaluations by bridging the gap between controlled experimental setups and the intricate nature of real-world language tasks. In this work, we extended the MQAR task by introducing variations in positional and n-gram patterns to investigate Mamba's underlying behavior. This analysis framework can also be adapted for evaluating other models, providing a broader tool for identifying similar issues.

A.2 Efficient Model Architecture

Efficient model architectures (*e.g., State-Space Models and Linear Attentions*) have become increasingly popular due to their ability to scale to long sequences while maintaining competitive performance. In this section, we focus on several key models that illustrate different strategies for improving efficiency.

S4 (Gu et al., 2021) introduced a novel class of sequence models designed to capture long-range dependencies using a state-space model (SSM) framework, typically formulated in continuous time. S4 introduces three key mechanisms to achieve this: (1) the Higher-Order Polynomial Projection Operator (HiPPO) (Gu et al., 2020), which efficiently memorizes signal history by operating on state and input transition matrices, (2) a diagonal plus low-rank (DPLR) parametrization that stabilizes the SSM matrix (A) by adding a low-rank correction, ensuring both diagonalizability and stability, and (3) efficient kernel computation through Fast Fourier Transforms (FFT) and inverse FFTs, reducing the overall complexity to $\mathcal{O}(N \log N)$. By leveraging these innovations, S4 significantly improves the handling of long-range dependencies, offering a more scalable alternative to traditional

models. The SSM parameters in S4 (Gu et al., 2022b) and S5 (Smith et al., 2023) are fixed after training, resulting in data-independent configurations that significantly limit the overall expressivity of models. In contrast, Mamba (Gu and Dao, 2023) addresses this limitation by introducing data-dependent parameters for S4.

Hyena (Poli et al., 2023) was designed to close the perplexity gap between attention-based Transformers and sub-quadratic alternatives. While traditional attention mechanisms face quadratic complexity with increasing sequence length, Hyena overcomes this by using implicitly parameterized long convolutions and data-gating, achieving sub-quadratic complexity. The model also shows that previous sub-quadratic attention approaches, such as those based on low-rank or sparse approximations, often still rely on dense attention layers to reach Transformer-level performance.

RWKV (Peng et al., 2023) is a recent RNN architecture designed specifically for language modeling, featuring a linear attention approximation mechanism called the "WKV" mechanism. RWKV utilizes linear time-invariant recurrences and can be viewed as the ratio of two state space models. Unlike traditional Transformers, which have quadratic complexity in terms of computation and memory, RWKV offers linear scalability, combining the efficiency of RNNs with the performance of Transformers. While RWKV is presented as a hybrid model of RNNs and Transformers, it primarily relies on linear attention and lacks the recurrent properties of traditional RNNs, making it more similar to attention-based models.

Based (Arora et al., 2024b) is a simple yet flexible architecture that integrates linear attention with sliding window mechanisms. By varying the window size and the feature dimension of the linear attention layer, Based can navigate the Pareto frontier of the recall-memory tradeoff. This allows it to effectively achieve full attention-like quality on one end while providing a compact state size for memory-efficient alternatives on the other.

GLA (Yang et al., 2024) introduces an efficient training algorithm for linear attention Transformers that integrates data-dependent gating mechanisms. This algorithm strikes a balance between floating-point operations (FLOPs) and parallelism, enabling the use of half-precision matrix multiplications to leverage modern GPU tensor cores. GLA exhibits competitive performance on language modeling tasks, demonstrating that gated linear attention Transformers can compete with strong baselines while ensuring computational efficiency.

Mamba-based Methods Mamba has recently garnered significant attention due to its efficient performance and ability to match or even surpass Transformers. However, some studies have highlighted its limitations in certain tasks, such as recall-based tasks, where its performance lags behind Transformer models. This limitation primarily stems from Mamba's fixed state size, which does not scale with input sequence length. To address this, Ben-Kish et al. (2024) proposed DeciMamba, a context-extension method aimed at improving Mamba's length generalization. This approach enables the model to effectively extrapolate to longer sequences without additional training, thereby mitigating the challenges associated with fixed state sizes. Additionally, Trockman et al. (2024) introduced a mimetic initialization technique, which optimizes the initialization of state-space model parameters to better mimic the behavior of attention-based models. This method enhances Mamba's recall capabilities by improving its ability to retain and retrieve long-range dependencies. Moreover, Wang and Li (2024) proposed StableSSM, a reparameterization technique designed to alleviate memory limitations in state-space models. By stabilizing the recurrent weights, StableSSM enhances the model's ability to capture long-term dependencies, thereby improving performance in tasks that require extended memory retention. Furthermore, Chen et al. (2024) identified a phenomenon termed "state collapse," where Mamba's performance degrades on sequences longer than those seen during training. To mitigate this, they proposed three techniques to enhance Mamba's length generalization, enabling it to process sequences beyond training length while preventing state collapse.

B Performance on Downstream Tasks

While the synthetic tasks provide valuable insights into how Global Gate mitigates shortcut behavior in Mamba, it is critical to assess how these improvements transform into real-world downstream tasks. For downstream evaluation, we use the LM evaluation harness from EleutherAI (Gao et al., 2021), following the approach of previous works. All models are trained on the same subset of the SlimPajama (Soboleva et al., 2023) dataset, using the *GPT-NeoX* tokenizer with a context length of 2048 for downstream tasks. We evaluate the fol-

Scale	Models	FDA	SWDE	SQUAD
130M Params	Pythia	28.22	20.24	12.23
3B tokens	Mamba	1.72	6.66	14.68
	w/ *GA*	2.45	7.11	16.66
340M Params	Pythia	55.26	61.57	22.82
15B tokens	Mamba	8.80	17.37	25.67
	w/ *GA*	9.53	19.98	26.44
790M Params	Pythia	62.96	66.14	28.16
30B tokens	Mamba	6.72	20.43	26.78
	w/ *GA*	8.17	21.78	30.13

Table 6: The results of models on three recall-intensive tasks, *GA* denotes the global gate strategy.

lowing tasks and datasets, which assess language modeling and common-sense reasoning capabilities:

- Wikitext (Merity et al., 2016)
- LAMBADA (Paperno et al., 2016)
- PIQA (Bisk et al., 2020)
- HellaSwag (Zellers et al., 2019)
- WinoGrande (Sakaguchi et al., 2021)
- ARC-challenge (Clark et al., 2018)
- ARC-easy: an easier subset of ARC-challenge
- OpenbookQA (Mihaylov et al., 2018)

Following the work of Arora et al. and Yang et al., we also evaluate our models on three recall-intensive tasks: FDA (Arora et al., 2024b), SWDE (Lockard et al., 2019), and SQUAD (Rajpurkar et al., 2018). These tasks focus on information extraction and reading comprehension, which can be viewed as real-world downstream tasks similar to MQAR. As shown in Table 6, while GA improves Mamba's performance across all tasks, subquadratic Mamba models significantly underperform attention-based model on both FDA and SWDE, which are information extraction tasks.

C Computational Cost

The original Mamba model utilizes a short convolution operation, whereas our proposed global gate strategy introduces an additional long convolution step. While this modification enhances the model's ability to capture long-range dependencies, it may also increase computational overhead.

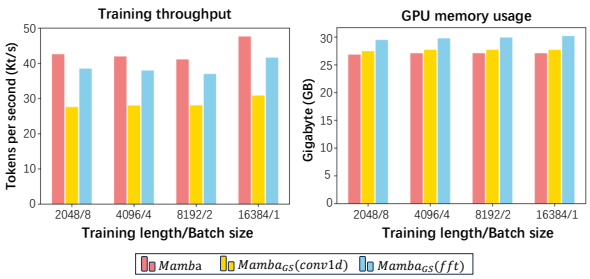


Figure 8: Training throughput and memory footprint of 130M scale Mamba on an Nvidia-A100 40GB GPU.

Figure 8 provides a detailed comparison of training throughput and GPU memory usage as a function of sequence length and batch size for different model configurations at the 130M model scale, evaluated on a single Nvidia A100-40GB GPU. **Mamba** refers to the original Mamba architecture, which uses the Hugging Face PyTorch implementation. **Mamba**$_{GA}$ (conv1d) and **Mamba**$_{GA}$ (fft) denote the Mamba model with the global gate strategy, where "conv1d" refers to the *nn.conv1d* implementation and "fft" refers to the *fast Fourier transforms* implementation.

In terms of training throughput, although the GS strategy incurs some loss in throughput, this degradation is relatively small and remains within an acceptable range, particularly when using the FFT-based implementation. As for GPU memory usage, the fft implementation results in higher memory consumption, while the *conv1d* implementation leads to a more modest increase in memory consumption Nevertheless, both implementations operate within reasonable memory limits, ensuring the practicality of the approach for large-scale models.

The original Mamba model is optimized with a hardware-aware algorithm, which is designed to accelerate on specific hardware platforms. Theoretically, our proposed long convolution step could be integrated into this process to further improve efficiency. However, this is not the focus of our current work, and we leave it for future research.

Further computational analysis at larger model scales is presented in Table 7. Each column represents different combinations of batch size and sequence length, with cell values indicating training throughput (tokens/s) and GPU memory usage (GB). The results are obtained using an FFT-based implementation of GA on a single Nvidia H20 GPU. While the global gate mechanism introduces some computational overhead, it effectively mitigates shortcut learning by enhancing the model's

Models	1-8192	2-4096	4-2048	8-1024
370M Mamba	19574 / 27.33	18049 / 27.33	18157 / 27.33	17965 / 26.64
+ GA	17559 / 31.72	16258 / 31.34	16400 / 31.20	16260 / 30.40
1.4B Mamba	7936 / 53.11	7414 / 53.11	7426 / 53.11	7364 / 50.53
+ GA	7007 / 62.64	6593 / 61.89	6620 / 61.60	6577 / 58.63

Table 7: Training throughput and memory footprint of larger scale Mamba and GA.

ability to leverage global information.

D Trade-off between Local and Global Gate

Performance on In-Domain Tasks Our experiments demonstrate that incorporating a **Global Gate (GA)** into Mamba enhances performance on various shortcut-related generalization challenges. However, for in-domain tasks that do not rely on long-range dependencies, introducing a global shortcut can slightly degrade performance.

Table 8 provides a detailed analysis of the trade-offs introduced by GA in in-domain tasks, particularly in scenarios where shortcut phenomena are less pronounced. The results indicate that incorporating global information primarily benefits shuffling-based tasks, likely due to the absence of identical patterns between training and testing phases. However, in most settings, GA leads to a slight performance drop. Notably, the *Last* setting inherently includes the *Std* configuration when the number of key-value pairs is sufficient to fill the entire context length, which explains the relatively strong performance of *Last-Std*. Meanwhile, the *Shuffle* setting represents the most generalized case, where performance tends to vary significantly. This effect is particularly pronounced in the N-gram Gathering setting, where fine-grained local information is critical. We hypothesize that while GA provides additional global context, it may also diminish the model's sensitivity to local details, thereby negatively impacting performance on tasks that rely heavily on fine-grained information.

Performance on K4V8-Shuffle Task In the K4V8-Shuffle variant, our experimental results show that the original Mamba struggles to perform effectively on this task. However, by introducing our proposed global gate strategy, we achieve a significant improvement in accuracy, boosting it from below 5% to 80%. The following case study, shown in Table 9, illustrates this enhancement. In this example, the original Mamba correctly predicts the first few tokens but fails to predict the entire value accurately. Since our evaluation method assesses the correctness of the entire output sequence, the original Mamba receives a low score. The issue stems from the limited convolution module in the original Mamba, where the convolution length is restricted to 4, insufficient to cover the key-value pairs in the K4V8 case. This limitation partially explains why the original model performs relatively better in K1V1 and K2V2 settings.

Performance on Real-World Downstream Tasks To illustrate the utility of global information processing in downstream tasks, we provide two examples where the original Mamba failed, but the GA model succeeded.

Example 1: Lambda Task

> *"She and Zach were covered in dust and sweat when Helen found them. 'Wow, Lexi! You rock.' Lexi groaned at the bad pun. Helen surveyed the work, which was nearly complete. 'How did you do this?' Lexi shrugged. 'Don't know.' 'It's her gift,' said __"*

The target entity in this example is Zach, which appears only at the beginning of the passage. Without global information processing, the model loses track of this reference in broader contexts, often misattributing the subject to more recently mentioned entities such as "Lexi" or "Helen." In contrast, the GA-enhanced model effectively retains long-range dependencies and correctly identifies "Zach."

Example 2: ARC-Challenge

> *"There are a total of eight planets that orbit the Sun. How many of the other planets orbit in the same direction as Earth?"*
>
> **Choices:** A. 0 B. 1 C. 4 D. 7

The correct answer is **D**. Answering this question requires the model to simultaneously consider information from the context ("a total of eight planets") and the question ("the other planets orbit in

Models	Standard	Last	Last-Std	Last-Shuffle	Shuffle	Shuffle-Std	Shuffle-Last	K1V1	K1V2	K2V2
Mamba	99.72	99.35	82.99	50.24	80.98	95.41	87.49	99.95	99.96	99.98
Mamba w/ *GA*	99.62	99.66	85.94	51.88	90.45	95.94	87.09	96.62	97.27	99.06

Table 8: Performance comparison of Mamba and Mamba with GA across different settings.

K4V8-Shuffle	
Key:	[1599, 7262, 5493, 4221]
Value:	[16301, 10098, 19263, 13834, 10106, 15846, 12555, 10962]
Mamba Prediction:	[16301, 10098, 19263, 13834, 12138, 11048, 10309, 19894]
Mamba GA Prediction:	[16301, 10098, 19263, 13834, 10106, 15846, 12555, 10962]

Table 9: A case of Mamba and Mamba with GA for the K4V8-Shuffle task. Value denotes the ground-truth tokens. The token highlighted in blue denotes the correct predictions.

the same direction as Earth"). The global information mechanism enables the model to capture and integrate these interdependent details across both the context and the question, leading to the correct response.

E Ablation Study

Effect of Global Window Sizes We conducted ablation studies on different global context window sizes using a synthetic task. Specifically, we experimented with Tiny Mamba using four different window sizes: 64, 128, 256, and 512. The results are presented in Table 10.

Model	Std-Std	Std-Last	Std-Shuffle
Mamba	99.03	12.07	14.31
GA-64	98.69	18.12	20.61
GA-128	98.50	15.81	22.46
GA-256	96.91	18.78	20.01
GA-512	97.95	14.20	20.70

Table 10: Performance of different global context window sizes on synthetic tasks.

While varying the global context window size leads to differences in performance, all configurations demonstrate a clear benefit in mitigating the shortcut problem compared to the original Mamba model.

Effect of Removing the Conv1D Layer We evaluated the impact of removing the Conv1D layer on synthetic tasks and observed that Mamba fails across all settings. This suggests that the absence of a token mixing mechanism significantly degrades performance.

Effect of Changing the Filter Size To further examine the role of the Conv1D layer, we experimented with different filter sizes and configurations, as shown in Table 11.

Configuration	Std-Std	Std-Last	Std-Shuffle
Replace with global convolution	99.03	22.70	31.56
Increasing filter size to 8	99.60	20.09	24.37
Increasing filter size to 16	99.57	22.93	27.10
Increasing filter size to 32	99.58	24.70	29.48
+ Global Gating mechanism	99.33	23.21	36.33

Table 11: Performance impact of different Conv1D filter sizes and modifications.

Increasing the filter size results in some performance improvements. However, we observed that merely expanding the original filter size is insufficient for handling more complex tasks, such as *k4v8shuffle*. This further supports our claim that Mamba requires effective modeling of both local and global information, where the Δ parameter plays a crucial role. Therefore, our modifications focus on optimizing Δ rather than modifying the Conv1D layer.

Attention-based Model Performance on MQAR Tasks We evaluate the performance of an attention-based model, i.e., Pythia-160M, on the experiments conducted in Section 3, aiming to verify whether transformer models exhibit similar shortcut phenomena as observed in Mamba.

Table 12 presents the performance of Pythia-160M under the Positional Pattern Change setting. We observe that regardless of the training mode, Pythia achieves near-perfect performance even when evaluated on test sets with positional patterns different from those seen during training. This indicates that, unlike Mamba, Pythia does not exhibit a pronounced shortcut phenomenon related to positional patterns. In the N-gram Gathering setting, Pythia exhibits a performance trend

Test \ Train	Standard	Last	Shuffle
Standard	99.98	99.04	98.80
Last	93.75	99.96	99.39
Shuffle	94.31	99.36	99.60

Table 12: Performance of Pythia-160M under the *Positional Pattern Change* setting.

Test \ Train	*K1V1*	*K1V2*	*K2V2*
K1V1	99.95	94.67	66.54
K1V2	0.00	99.97	85.20
K2V2	0.00	1.21	100.00

Table 13: Performance of Pythia-160M under the *N-gram Gathering* setting.

similar to Mamba. The most notable case arises when the model is trained on $K1V1$ and tested on $K2V2$. While Mamba achieves near-perfect performance in this scenario, Pythia completely fails. This observation aligns with our previous hypothesis: Mamba's success is likely not due to true n-gram recall but rather an over-reliance on structural cues provided by special characters or templates. This hypothesis is further supported by its performance on the K2V2-Robustness task, as depicted in Figure 9. Notably, compared to Mamba, Pythia is significantly less affected by noise in the key tokens. This suggests that in the $K2V2$ scenario, Pythia primarily relies on both key tokens for decision-making.

F Standard Deviation on Syntactic and Downstream Tasks

Our observations reveal that Mamba's training is highly unstable and often prone to overfitting. Notably, the Mamba architecture lacks stochastic modules such as dropout, indicating that the instability may be inherent to the model's architecture. Since it is not the focus of this work, we leave further exploration for future work.

For synthetic tasks, we fixed the random seed to 42 during the training process. Additionally, when constructing synthetic data, we assigned different random seeds for different settings and data splits. This approach ensures that there are no overlapping key-value pairs between the training and testing datasets, allowing for a more accurate evaluation of the model's recall capabilities. we trained and evaluated the 13M model using multiple random seeds to assess variability (due to computational

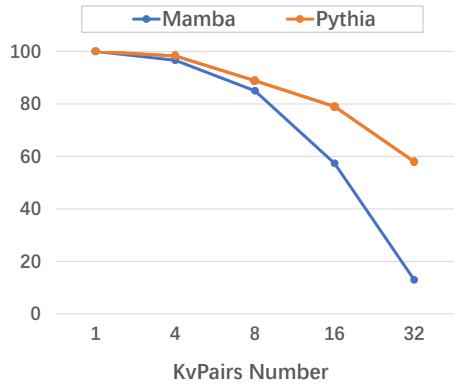


Figure 9: Comparison between Mamba and Pythia on the *K2V2-Robustness* setting.

Model	Std-Std	Std-Last	Std-Shuffle
Mamba	98.95 $\pm$0.004	17.45$\pm$0.013	20.79$\pm$0.010
w/ *GA*	99.25$\pm$0.068	23.17$\pm$0.011	26.33$\pm$0.007

Table 14: Standard deviation results for the Mamba model on position change tasks.

constraints, larger-scale models were too costly). The results are depicted in the Table 14.

For downstream tasks, we provide the detailed training recipe in Appendix B. The evaluation is conducted using the lm-evaluation-harness from EleutherAI. For these zero-shot tasks, conducting multiple tests with different random seeds does not result in significant variations.

G Detail for MQAR and its vanriants

This section provides additional details for the analysis experiments discussed in Sec. 3.

G.1 General Data Configuration

We utilize mixed input sequence lengths and varying key-value pair counts for the MQAR tasks. The parameters L (sequence length), N (number of key-value pairs), and α are employed to adjust these properties. Unless specified otherwise, the vocabulary size is set to 20,000. The training set consists of sequence lengths $L \in \{64, 128, 256, 512\}$, with the corresponding number of key-value pairs N satisfying $2^N \leq \frac{L}{2}$. For empty positions, we insert random values as padding. For each configuration, there are 10,000 examples in the training set and 100 examples in both the validation and test sets. All other settings are consistent with the original MQAR task described in (Arora et al., 2024a).

G.2 Positional Pattern Change

We categorize position patterns into three distinct types: standard MQAR, last, and shuffle. For the standard MQAR pattern, we follow Arora et al., where the key-value pairs are clustered at the beginning of the sequence, and queries are randomly scattered in the remaining positions. In the last pattern, we divide the input sequence into two equal-length segments, placing all key-value pairs in the last portion of the first segment, while queries are randomly distributed in the second segment. For the shuffle pattern, we again split the entire input sequence into two equal parts, but we randomly place key-value pairs in the first half, with queries randomly scattered in the second half. Other settings remain consistent with the general configuration.

G.3 N-gram Gathering

We refer to the original MQAR task as the k1v1 pattern. To assess the model's capability and its tendency to rely on shortcuts in n-gram patterns, we increase the number of tokens in both the keys and values. In this setup, to ensure that the model can effectively distinguish between the key and the value, we introduce special separators between token groups in both the key and value sequences.

Specifically, a key-value pair in the *K2V2* pattern appears as: <SEP> k_0 k_1 <SEP_KV> v_0 v_1 <SEP> Although the addition of special symbols may increase the length of individual key-value pairs and cause the model to focus on these symbols, it is necessary. Without the special symbols, a key with N tokens would be indistinguishable from a key with just 1 token, as the model could rely solely on the last token in the N-token key to retrieve the corresponding value. Similarly, cases involving M-token values could often be reduced to multiple instances of $M-1$ 2-gram patterns, which would fail to verify if the model truly follows the n-gram relationship. Thus, including these special characters is crucial. It is also important to note when the value consists of more than one token, accuracy is measured based on the entire value sequence. In other words, all M tokens of the value must be predicted correctly for the key-value pair to be considered correct.

G.4 Noise Injection

We test the robustness of the two previously mentioned shortcut patterns. For position robustness, we divide the input sequence into two equal-length segments. Then, we further split the first half into N sections by position. In our experiments, we set $N = 4$. In each of these N sections, we insert the same key, each corresponding to a different value. The same key is then placed as a query in the second half of the sequence. The model is considered correct if it recalls any of the corresponding values. This setup allows us to assess whether the model, trained on different position patterns, exhibits any positional bias when recalling key-value pairs—specifically, whether it tends to favor key-value pairs from certain positions.

In this configuration, the test data consists of sequence lengths $L \in \{64, 128, 256\}$ and corresponding key-value pairs $N \in \{8, 16, 32\}$, with 100 examples per configuration.

For n-gram robustness, we introduce noisy keys by fixing the last $n-1$ tokens of the n-token key while randomly replacing one token. This creates n-gram noise. Specifically, in the k2v2 setup, we fix the second position in the key, and the number of noisy key-value pairs matches the number of original key-value pairs.

G.5 Training and Evaluation Configuration

For all synthetic runs on MQAR and its variants, we apply the following training protocol:

- **Optimizer and Schedule**: We use weight decay of 0.1, a warmup duration of 10%, and a linear warmup schedule. The AdamW optimizer is employed. For each setting, we sweep the learning rates lr $\in$ {3e-3, 1e-3, 8e-4, 5e-4, 1e-4}. All models are trained for 30 epochs without early stopping with 8×A100-40GB GPU.

- **Data**: Each model is trained and evaluated on 10,000 training examples and 100 validation examples per setting. The random seed for each configuration is fixed, and the data, as well as its order, remains constant across all runs.

- **Loss**: During training, we calculate the cross-entropy loss (CELOSS) at all value positions corresponding to each query. For n-gram setups, where special tokens are present, we additionally compute the loss for the last <SEP> token. We found this helps the model understand the role of special tokens in segmenting the sequence.

Evaluation: We evaluate the model using the checkpoint that performs best on the validation set. The training, validation, and test data are constructed in the same way, with identical input sequence lengths and numbers of key-value pairs, differing only in the number of data points and random seeds. However, specific settings are applied for the robustness experiments.

A case is considered correct only if all key-value pairs are predicted correctly. In multi-value setups (KNVM), all M values corresponding to a key must be predicted accurately to be considered correct. The final results are reported as the average across all data points.

H Detail for Model

For our experiments, we adopt the standard Mamba-130M configuration from (Gu and Dao, 2023), as smaller models tend to struggle with the MQAR tasks and do not accurately reflect the shortcut phenomenon. All models are trained on the same subset of the SlimPajama (Soboleva et al., 2023) dataset with the *GPT-NeoX* tokenizer for downstream tasks. All models are trained with AdamW (Loshchilov and Hutter, 2019) using a maximum learning rate of 8e-4. All models are trained on 3B tokens with a batch size of about 0.5M tokens. We use a cosine learning rate schedule with a warmup of 10% steps. We use a weight decay of 0.01, and gradient clipping of 1.0.

The settings for other baselines are as follows:

- **Pythia-133M**: We reduce the number of layers to 8 while keeping other configurations consistent with Pythia-130M (Biderman et al., 2023). This adjustment minimizes the number of parameters to ensure a fair comparison in our analysis.
- **Hyena-153M**: The configuration matches that of Hyena-153M.
- **RWKV-153M**: We set d_{model} to 1024 and the number of layers to 12, maintaining other configurations consistent with RWKV-Pile-430M.
- **Mamba-130M**: We adopt the standard configuration from Gu and Dao.
- **Mamba-340M**: We adopt the standard configuration from Gu and Dao.
- **Mamba-790M**: We adopt the standard configuration from Gu and Dao.
- **Mamba-1.4B**: We adopt the standard configuration from Gu and Dao.
- **Mamba-mimetic_init**: Trockman et al. introduced a mimetic initialization technique to enhance Mamba's recall ability. Since their code has not been released, we use our own implement based on the paper, ensuring $A, \Delta \approx 1$ and $W_T^C W_B \approx I$. Specifically, we parameterize A as $A = -\exp(-c_A \log)$ with $c = 8$, making $A_d^T \Delta_d \approx 0$ in Eq. 7. Additionally, we set $W_\Delta \approx 0$ and $b_\Delta = \text{softplus}^{-1}(1) \approx 0.54$, ensuring $\Delta_d \approx 1$.
- **Mamba-*state_size***: We modify the *ssm_state_size* of the Mamba model, which is originally set to 16 in the 130M configuration, while keeping all other settings consistent with Gu and Dao.
- **Mamba-*Global_Selection***: We define Δt as the key to Mamba's selectivity. However, it is constrained by the short convolution and linear transformation. To address this, we propose incorporating more global information into Δt. Specifically, we introduce a long convolution module to model distant historical context and use its result to gate the original Δt. This is formulated as follows:

$$\begin{aligned}\Delta_t = \tau((\mathbf{W}_2 \cdot \sigma\left(\mathbf{W}_1 \cdot \text{Conv}_{\text{short}}(\mathbf{X}_t)\right) \\ \odot \sigma\left(\text{Conv}_{\text{long}}(\mathbf{X}_t)\right))\end{aligned} \tag{8}$$

where Conv denotes the convolution operation, τ denotes the softplus function, $\mathbf{W}$ represents the linear transformation operations, σ is the nonlinear activation function, specifically the Silu (Ramachandran et al., 2017), and $\odot$ is the element-wise multiplication operation. The kernel size for the short convolution is set to 4, following (Gu and Dao, 2023), while the kernel size for the long convolution is set to $\frac{1}{4}$ of the input sequence length for syntactic tasks and the full input sequence length for downstream tasks. For models with 790M and 1.4B parameters, we employ an FFT-based implementation, whereas smaller-scale models use a `conv1d`-based implementation. In the analysis experiments, we add a global gate module to every layer of Mamba, as the relatively short input length ensures that the long

convolution does not introduce excessive parameters. However, in downstream tasks, we only incorporate global gate parameters in certain layers to maintain fairness in model parameter counts.

I More Analysis on the Selectivity of Mamba

In this section, we provide further analysis to supplement the discussion from the previous section.

Theoretical analysis of Mamba on MQAR Based on the recurrence equation, the hidden state at each step is updated as:

$$h[i,j] = A_i\, h[i-1,j] + B_i\, u[i,j], \qquad (9)$$

where the following components must be stored: the previous hidden state $h[i-1,:] \in \mathbb{R}^d$ and the parameters $\Delta_i, B_i, C_i \in \mathbb{R}^d$, which are derived from the input sequence $u[0 \ldots i-1]$. Consequently, the storage requirement for each hidden state Z_i^{Mamba} is $Z_i^{Mamba} \in \mathbb{R}^{4d}$, leading to an overall storage complexity of $O(N \cdot d)$.

Arora et al. (Arora et al., 2024b) provide a rigorous analysis using randomized communication complexity by reducing an autoregressive (AR) task to the index problem. In the index problem, Alice holds a binary string $x \in \{0,1\}^N$ and Bob holds an index $i \in [N]$; to correctly recover x_i, any one-way (causal) model must effectively "remember" nearly all N bits of the input. This implies that, for causal recurrent models, the hidden state must have a capacity of at least $\Omega(N)$ bits, regardless of the nominal model dimension d. In other words, even if each parameter is stored using $O(\log N)$ bits, the minimal effective storage per hidden state must scale as

$$\dim(Z_i^{Mamba}) \geq \Omega(N). \qquad (10)$$

Moreover, when global information—such as that provided by long convolutions—is introduced to capture full-sequence context, the storage requirement per hidden state increases from $O(d)$ to $O(d+N)$. This extra $O(N)$ term is necessary to satisfy the communication complexity lower bound and to correctly model long-range dependencies. Thus, in a purely causal setting, the overall storage complexity becomes $O(N \cdot (d+N))$, which, when N dominates, is $O(N^2)$.

These theoretical conclusions not only match the lower bounds derived via communication complexity (by reduction to the index problem) but also motivate recent architectural innovations. For example, approaches such as "Just Read Twice" prompting and non-causal recurrent architectures aim to mitigate the full $\Omega(N)$ memory requirement by reordering the input so that only the smaller of two sets (when the input is naturally divided into parts) needs to be retained in memory. However, in the standard Mamba model, which processes input causally, the necessity to store $\Omega(N)$ bits per hidden state remains, leading to the $O(N^2)$ overall storage complexity and emphasizing the inherent tradeoff between memory efficiency and recall capability in AR tasks.

Not all layers are selective: We first examine the attention behavior of each Mamba layer trained on the standard MQAR task and evaluate the model's performance on the same pattern. The attention matrices for all layers of Mamba on the standard MQAR task are plotted in Figure 13. One notable observation is that many layers exhibit attention scores concentrated around the main diagonal, while other regions are largely inactive (black). We consider these layers as primarily responsible for organizing and propagating the hidden state, rather than making specific key-value selections. In contrast, certain layers demonstrate more pronounced selectivity, such as layer 22 in Figure 13, where the attention distribution sharply focuses on the key-value pair regions. Clearly, these selective layers are crucial to Mamba's ability to identify relevant information. Therefore, our analysis primarily focuses on these layers.

Perfectly Selective on In-domain Patterns: Mamba performs exceptionally well on the standard pattern of the MQAR task. We investigate whether Mamba's perfect performance on these patterns results from its correct selective capabilities. As shown in Figures 10 and 11, these figures depict Mamba's attention distribution under varying input sequence lengths and different numbers of key-value pairs. It can be observed that, regardless of changes in input length or key-value pair quantity, Mamba consistently captures the correct key-value positions, enabling accurate retrieval of the correct answers. This observation strongly suggests that the accuracy of the MQAR task is highly dependent on Mamba's ability to capture the correct key-value pairs. If the model can accurately

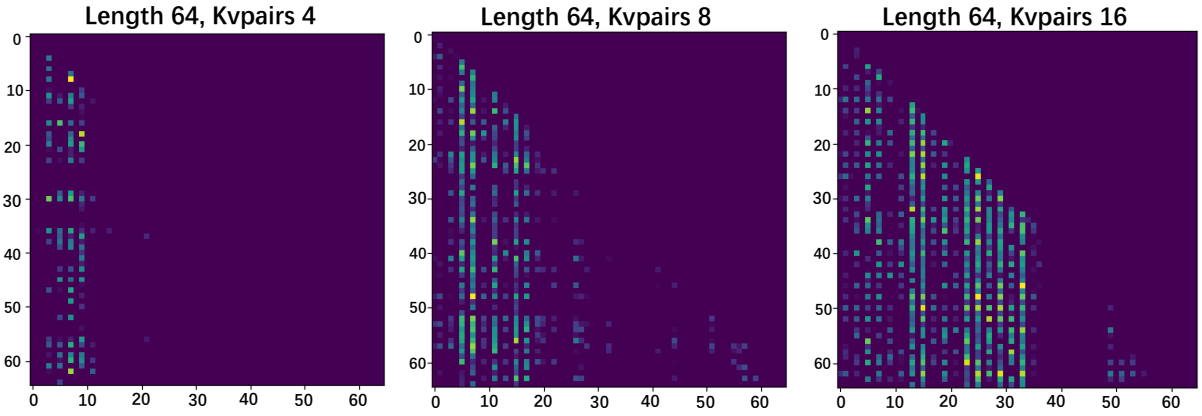


Figure 10: The attention matrices of Mamba on MQAR tasks, where the input sequence length is 64, and the number of key-value pairs varies from 4 to 16. Lighter colors indicate higher attention weights at specific positions. The results are from the 22st layer of the Mamba model.

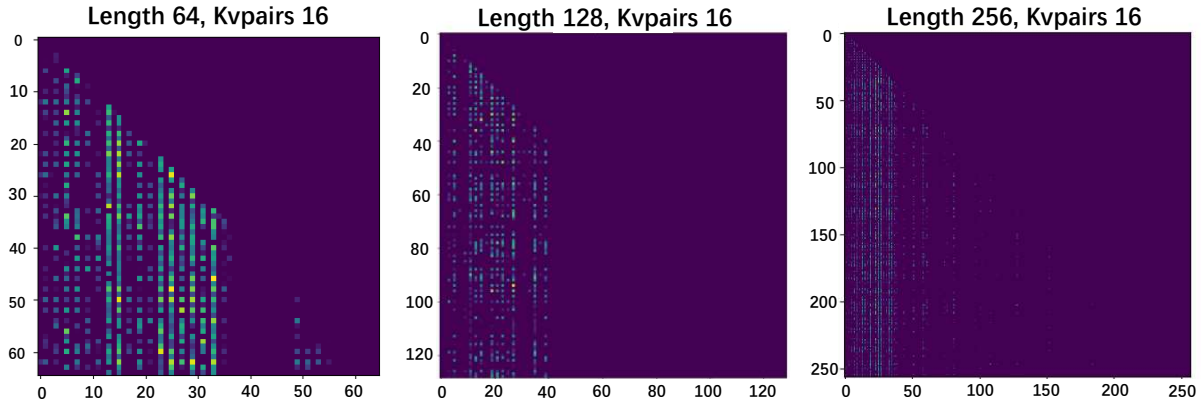


Figure 11: The attention matrices of Mamba on MQAR tasks, where the number of key-value pairs is 16, and the input sequence length varies from 64 to 256. Lighter colors indicate higher attention weights at specific positions. The results are from the 22nd layer of the Mamba model.

pinpoint the positions of these pairs, it has a high probability of correctly retrieving the corresponding value for any given key.

The Role of Δ_t in Selectivity: In Mamba, the parameters A and B play a crucial role in balancing the model's focus between historical information and new inputs. A key component for this process in common is Δ_t, which directly governs the model's selective behavior. By adjusting Δ_t, Mamba determines whether to prioritize recent inputs or maintain the influence of past states.

The parameter Δ_t functions similarly to gating mechanisms in recurrent neural networks (RNNs). When Δ_t is large, the model resets its internal state, focusing on the current input and diminishing the impact of previous information. In contrast, a small Δ_t preserves the existing state, allowing the model to ignore the current input and continue prioritizing historical context. This behavior parallels the function of gates in RNNs, where a larger gate value highlights new information, and a smaller gate value retains past states.

Mamba can be interpreted as a state-space model (SSM), where Δ_t controls the degree of continuity in the system. A large Δ_t indicates that the model focuses on the current input for an extended period, effectively discarding older information, while a small Δ_t implies that the input is transient and has a reduced effect on the model's decision-making process. This mechanism enables Mamba to adapt its selective attention dynamically based on the requirements of the task at hand.

J Performance on Standard MQAR

Prior research (Arora et al., 2024a,b) has proven that *attention-based* models surpass *rnn-based* models on MQAR task. MQAR requires models to memorize key-value pairs in their hidden

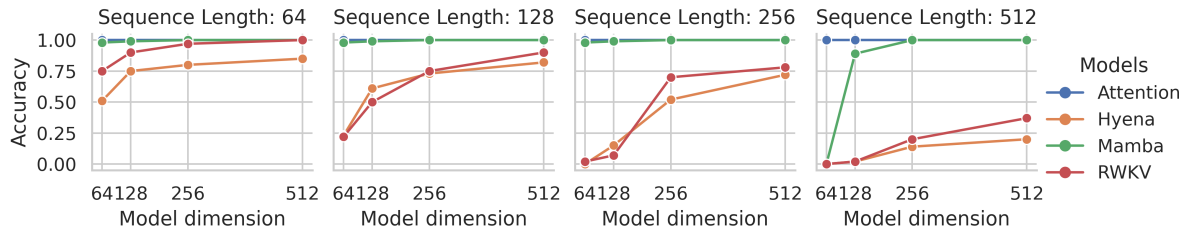


Figure 12: Performance of various models, including H3 (Fu et al., 2023), Hyena (Poli et al., 2023), and RWKV (Peng et al., 2023), alongside *attention-based* models (Touvron et al., 2023), across different model dimensions on MQAR tasks. The input sequences vary from 64 to 512 tokens. Other experimental settings are consistent with Arora et al..

state, which presents a significant challenge for *rnn-based* models, as they maintain a fixed-size state to handle all historical information. 12 depicts performance of different models on the standard MQAR task settings.

It is evident that *rnn-based* models significantly lag behind *attention-based* models, particularly as the length of the input sequence increases. Attention mechanisms, characterized by their quadratic token interactions, excel in identifying key tokens within the MQAR task and extracting the corresponding values. Such architectures can utilize the entire past sequence as memory, enabling effective retrieval of prior information and precise recall of associated values. In contrast, *rnn-based* models are required to store all past information within a single hidden state, relying on this compressed memory to retrieve the relevant keys and values based on the current query. As the sequence length increases, the compressed historical information grows, making it harder to retrieve the corresponding key-value pairs, leading to worse performance. An exception to this trend is Mamba, which consistently exhibits performance comparable to attention mechanisms across most settings and significantly outperforms its *rnn-based* counterparts at equivalent model dimensions. Prior works attribute Mamba's success to its data-dependent features. By incorporating input-dependent matrices $\overline{A}$ (via discretization), $\overline{B}$, and $\overline{C}$, Mamba effectively retains essential details while discarding irrelevant information. However, our experiments reveal that this success may not be as intentional or precise as it initially seems. Mamba's success partly results from its heavy reliance on superficial, local patterns (e.g., attention to the beginning of the input), which we defined as ***"local pattern shortcuts"*** rather than a genuine ability to recall.

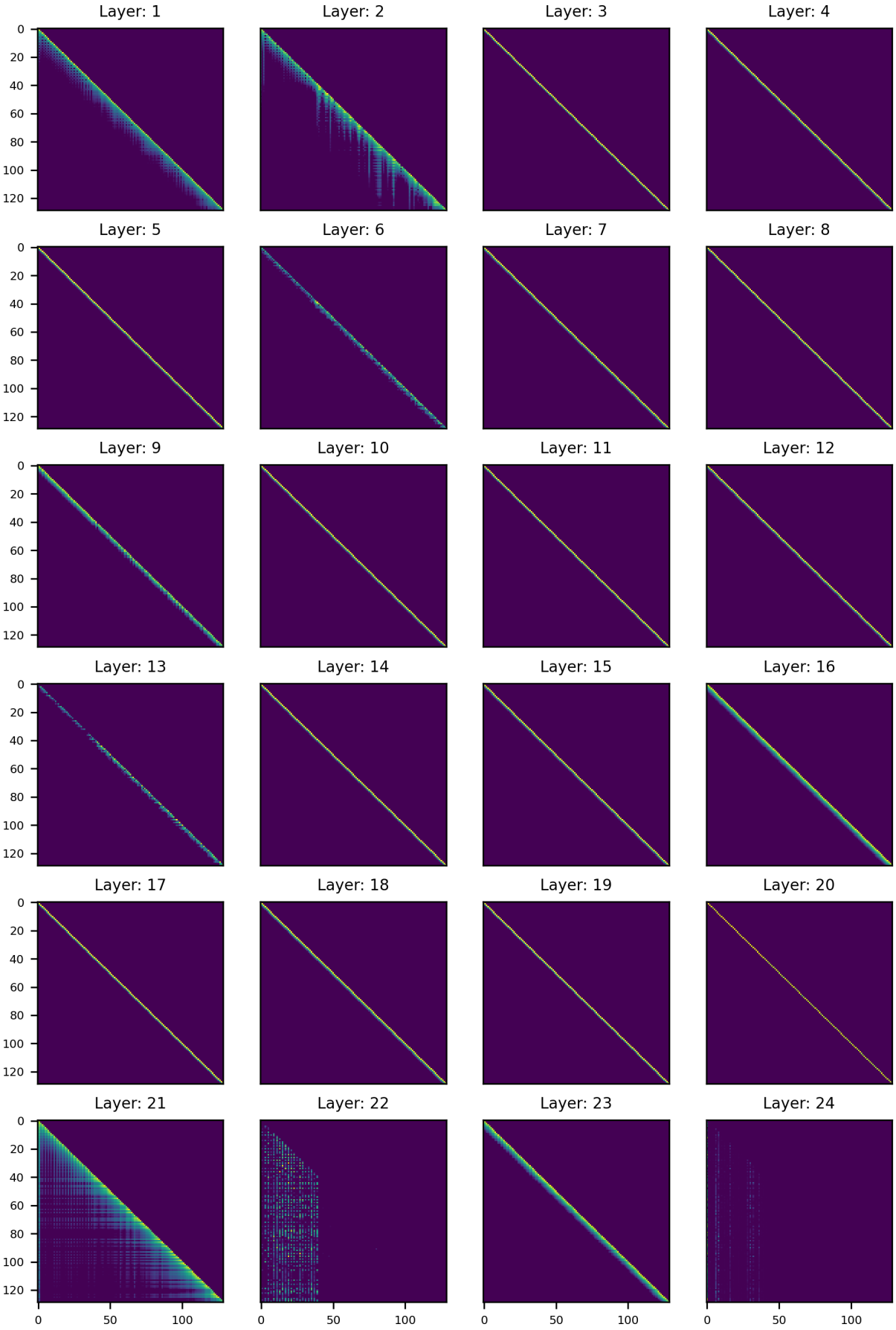


Figure 13: The attention matrices of all layers of Mamba on MQAR tasks, where the input sequence length is 128, and the number of key-value pairs is 16. Lighter colors indicate higher attention weights at specific positions.