EFFECTS OF INPUT VARIABILITY ON LONG-TERM DEPENDENCIES IN SELECTIVE STATE SPACE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

State space models (SSMs) have attracted attention for their memory and computational efficiency in modeling long sequences. In particular, Mamba enables modeling of time-varying dynamics through the selective SSM (S6), an input-dependent mechanism, while preserving parallel computation. Despite its empirical success, compared to input-independent SSMs such as S4 and S5, Mamba may underperform on long-sequence tasks that require preserving information regardless of subsequent inputs. Here we analyze how input variability affects long-term dependencies in the S6 recurrence. We show that larger input variance can make it difficult to preserve previous information over long sequences, suggesting that the discretized state matrix should contain values close to 1 at initialization. We further show that under the standard Mamba initialization, increasing the state size shifts the discretized state matrix distribution toward values close to 0, without increasing the proportion of values close to 1. These findings lead to a simple initialization approach based on the Gumbel-Softmax trick, which addresses the issues identified in our analysis by modifying the initialization distribution without any architectural changes. We empirically validate the derived initialization properties on synthetic and LRA tasks, showing that the proposed initialization improves performance on tasks that require preserving prior information over long sequences regardless of subsequent inputs, while maintaining comparable performance on context-dependent tasks. These results help explain why input variability can degrade long-term dependencies in selective SSMs.

1 INTRODUCTION

Linear recurrent neural networks (RNNs) have attracted attention because of their memory and computational efficiency as an alternative to Transformers. Linear RNNs have linear dependencies on previous hidden states, which enables us to use parallel computation, such as convolution and parallel scan algorithms, improving the computational efficiency of RNNs. To date, various linear RNN variants have been proposed (Gu et al., 2022a;b; Smith et al., 2023; Orvieto et al., 2023; Qin et al., 2023), demonstrating the efficacy on modeling long sequences on benchmarks. Notably, Structured State Space Models (SSMs) (Gu et al., 2022a) achieve both memory efficiency and linear time complexity, which improves modeling of long sequence tasks, as efficient alternatives to Transformers. Since SSMs cannot adapt their dynamics depending on the input at each time step, Mamba has been introduced (Gu & Dao, 2024; Dao & Gu, 2024), which introduces an input-dependent mechanism using the selective state space model (S6) to incorporate time-varying dynamics while achieving parallel computation.

Despite the empirical success of Mamba, it has been reported to underperform on *long-sequence modeling tasks* in the Long Range Arena (LRA) benchmark (Tay et al., 2021), particularly on Image and Pathfinder. These tasks require retaining information over long sequences, where the relevance of past inputs cannot be inferred from the current input alone. For example, the Pathfinder task may require the model to preserve the initial state until the end of the sequence regardless of the inputs. For such settings, initialization can play an important role. If values close to 1 are insufficient at initialization, learning to acquire them from the inputs can be difficult, making long-term dependencies hard to capture. This perspective is consistent with prior studies on gated RNNs (Tallec & Ollivier, 2018; Gu et al., 2020), which show that initializing forget-gate biases close to 1 is effective for learning long-term dependencies.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

In this paper, we analyze how input variability affects long-term dependencies in selective state space models. We focus on the S6 recurrence, the core component of Mamba. In S6, the input not only updates the hidden state, but also changes the discretized state matrix depending on the inputs. Thus, subsequent inputs can accelerate the decay of previously stored information that should be preserved regardless of the inputs. Our analysis shows that larger input variance decreases the ability to preserve previous information. This suggests that Mamba may underperform on tasks that require preserving previous information regardless of subsequent inputs, while remaining useful for context-dependent tasks such as language modeling. This analysis suggests that the discretized state matrix should take values close to 1 at initialization, since such values are insensitive to input variance and thus preserve previous information over long sequences. This analysis also suggests that making the bias sufficiently negative pushes the discretized state matrix toward values close to 1, which can improve long-range retention even under high input variance. In contrast, pushing all values close to 1 makes the discretized state matrix insensitive to the input, making it difficult to flexibly adjust the discretized state matrix through inputs.

We can summarize two initialization requirements from the above observations: The discretized state matrix should contain values close to 1 to retain information, while also containing middle-regime values for input-dependent control. Thus, long-term dependencies and input sensitivity should be balanced at initialization, rather than optimized by pushing all values close to 1. We also show that the standard Mamba initialization does not explicitly satisfy these requirements. Although the initialized discretized state matrix contains a broad range of timescales from 0 to 1, increasing the state size shifts its distribution towards values close to 0, without increasing the proportion of values close to 1.

Based on our analysis, we introduce a simple bias initialization based on the Gumbel-Softmax trick (Maddison et al., 2017; Jang et al., 2017; Li et al., 2018), which increases the ratio of values close to 1 while preserving input-sensitive middle-regime values at initialization. This initialization includes the standard Mamba initialization as the special case. We empirically validate our analysis on synthetic copying and selective copying tasks, and on LRA tasks. The proposed initialization improves performance on tasks that require preserving information regardless of subsequent inputs (Image, Pathfinder), while maintaining performance on context-dependent tasks (Text). Our goal is to understand one mechanism behind the degradation of S6 on long-sequence tasks, not to achieve state-of-the-art performance. Unlike architecture-level approaches (Ben-Kish et al., 2025; Yu & Erichson, 2025), our method modifies only the bias initialization and leaves the architecture unchanged.

Our contributions are summarized as follows:

- We identify that input variability in S6 can degrade long-term dependencies and characterize the importance of values close to 1 and middle-regime values.
- We show that the standard Mamba initialization causes an unintended distribution shift depending on the state size, which can fail to balance these requirements.
- We use a simple bias-initialization based on the Gumbel-Softmax trick to verify our analysis without changing the S6 architecture and confirm that the derived initialization improves performance on tasks that require capturing long-term dependencies regardless of subsequent inputs, while preserving performance on context-dependent tasks.

2 PRELIMINARIES

Structured state space models (S4). *Structured state space models* (S4) (Gu et al., 2022a) are variants of state space models (SSMs) that address the limitation of parallelizability in traditional RNNs, achieving both memory efficiency and linear time complexity. We first consider continuous-time linear SSMs. Given an input signal $\mathbf{x}(t) \in \mathbb{R}^d$, a latent state $\mathbf{h}(t) \in \mathbb{R}^N$, and an output signal $\mathbf{y}(t) \in \mathbb{R}^d$, a continuous-time linear SSM is defined as

$$\frac{d\mathbf{h}(t)}{dt} = \mathbf{A}\mathbf{h}(t) + \mathbf{B}\mathbf{x}(t), \quad \mathbf{y}(t) = \mathbf{C}\mathbf{h}(t) + \mathbf{D}\mathbf{x}(t),$$

where $\mathbf{A} \in \mathbb{R}^{N\times N}$, $\mathbf{B} \in \mathbb{R}^{N\times d}$, $\mathbf{C} \in \mathbb{R}^{d\times N}$, and $\mathbf{D} \in \mathbb{R}^{d\times d}$.

To process a discretized input signal, treated as a sequence $\mathbf{X} = (\mathbf{x}_0, \ldots, \mathbf{x}_T)$ with $\mathbf{x}_t \in \mathbb{R}^d$, S4 models discretize the continuous-time SSM into the following linear recurrence:

$$\mathbf{h}_t = \bar{\mathbf{A}}\mathbf{h}_{t-1} + \bar{\mathbf{B}}\mathbf{x}_t, \quad \mathbf{y}_t = \bar{\mathbf{C}}\mathbf{h}_t,$$

where $\bar{\mathbf{A}}$ is a discretized state matrix, $\bar{\mathbf{B}}$ is a discretized input matrix, and $\bar{\mathbf{C}} \in \mathbb{R}^{d\times N}$ is a trainable parameter. For simplicity, we omit the parameter $\mathbf{D}$, as in Gu et al. (2022a). For a constant step size Δ, the discretized state matrix $\bar{\mathbf{A}}$ and the discretized input matrix $\bar{\mathbf{B}}$ are computed with zero-order hold (ZOH) using the following equation:

$$\bar{\mathbf{A}} = \exp(\Delta\mathbf{A}), \quad \bar{\mathbf{B}} = (\Delta\mathbf{A})^{-1}(\exp(\Delta\mathbf{A}) - \mathbf{I})\Delta\mathbf{B}.$$

Selective state space model (S6). The *selective state space model* (S6) (Gu & Dao, 2024) is a variant of SSMs and the core component of Mamba. S6 can dynamically optimize its outputs depending on inputs because its discretized state matrices are computed as follows:

$$\mathbf{S}_t = \bar{\mathbf{A}}_t \odot \mathbf{S}_{t-1} + \mathrm{diag}(\mathbf{x}_t)\bar{\mathbf{B}}_t, \quad \mathbf{y}_t = \mathbf{S}_t\mathbf{C}_t,$$

where $\odot$ denotes elementwise multiplication, $\mathbf{S}_t \in \mathbb{R}^{d\times N}$ is the hidden-state matrix, and $\mathbf{S}_{t-1} \in \mathbb{R}^{d\times N}$ is the previous hidden-state matrix. The discretized state matrix $\bar{\mathbf{A}}_t$ and the discretized input matrix $\bar{\mathbf{B}}_t$ are computed as follows:

$$\bar{\mathbf{A}}_t = \exp(\mathrm{diag}(\boldsymbol{\Delta}_t)\mathbf{A}), \quad \bar{\mathbf{B}}_t = \boldsymbol{\Delta}_t \otimes (\mathbf{B}\mathbf{x}_t), \quad \mathbf{C}_t = \mathbf{C}\mathbf{x}_t, \quad \boldsymbol{\Delta}_t = \mathrm{softplus}(\mathbf{W}_\Delta\mathbf{x}_t + \mathbf{b}_\Delta),$$

where $\mathbf{A} \in \mathbb{R}^{d\times N}$, $\mathbf{B} \in \mathbb{R}^{N\times d}$, $\mathbf{C} \in \mathbb{R}^{N\times d}$, and $\mathbf{W}_\Delta \in \mathbb{R}^{d\times d}$ are trainable parameters, $\mathbf{b}_\Delta \in \mathbb{R}^d$ is a bias vector, and $\otimes$ denotes the outer product. Unlike the constant step size Δ, $\boldsymbol{\Delta}_t \in \mathbb{R}^d$ is input-dependent. We denote its i-th component by $\Delta_{t,i} = \mathrm{softplus}(\mathbf{w}_i^\top\mathbf{x}_t + b_{\Delta,i})$, where $\mathbf{w}_i^\top$ is the i-th row of $\mathbf{W}_\Delta$.

Initialization of Mamba parameters. In the official Mamba implementation (Gu & Dao, 2024), the state matrix $\mathbf{A}$ and bias term $\mathbf{b}_\Delta$ are initialized as follows:

$$\begin{aligned} \mathbf{A}_{i,j} &= -j \quad (i = 1, \ldots, d,\ j = 1, \ldots, N), \\ \gamma &= \exp\Big(u\big(\log\Delta_{\max} - \log\Delta_{\min}\big) + \log\Delta_{\min}\Big), \\ b_\Delta &= \mathrm{softplus}^{-1}(\gamma), \end{aligned} \tag{1}$$

where $u \sim \mathcal{U}[0,1]$ is sampled from the uniform distribution, and $\Delta_{\max}, \Delta_{\min}$ are maximum and minimum values of initialization Δ. Due to this implementation of $\mathbf{A}$ and $\boldsymbol{\Delta}$, discretized state matrix distributions change drastically depending on the state size.

3 Investigation for the initial distribution of discretized state matrix on S6

We illustrate the distribution of the discretized state matrix shifts with respect to changes of the state size under the Mamba initialization. We first analytically characterize it in Section 3.1, followed by empirically observing distribution shifts in Section 3.2.

3.1 Initial discretized state matrix distributions under the Mamba initialization

Under the Mamba initialization $A_{i,j} = -j$ $(j = 1, \ldots, N)$, for the i-th row $\mathbf{w}_i^\top$ of $\mathbf{W}_\Delta$, we have $\Delta_{t,i} = \mathrm{softplus}(\mathbf{w}_i^\top\mathbf{x}_t + b_{\Delta,i}) > 0$. Since the discretized state matrix for each state index j is given as $\bar{A}_{t,i,j} = \exp(-j\,\Delta_{t,i})$, the following holds: For any threshold $c \in (0,1)$ and $\Delta_{t,i} > 0$, the number of state indices satisfying $\bar{A}_{t,i,j} \geq c$ satisfies

$$\big|\{j \in \{1, \ldots, N\} : \bar{A}_{t,i,j} \geq c\}\big| \leq \left\lfloor \frac{\log(1/c)}{\Delta_{t,i}} \right\rfloor. \tag{2}$$

That is, under the Mamba initialization, the number of values close to 1 in $\bar{A}_{t,i,j}$ does not increase as N increases, while the number of values close to 0 grows with N.

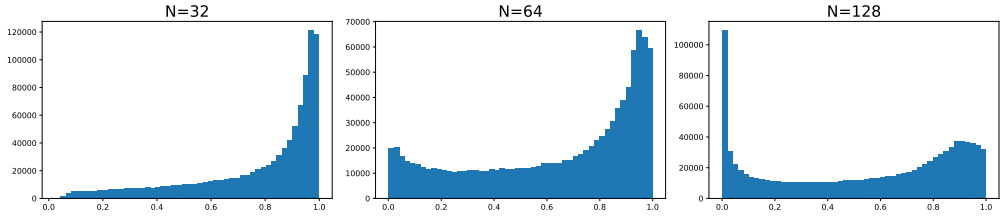


Figure 1: Discretized state matrix distributions for state sizes of 32 (left), 64 (center), and 128 (right) of S6 on the Image task. Increasing the state size does not increase the fraction of values close to 1.

3.2 Empirical observations of initial discretized state matrix distributions on real-world data

We empirically illustrate the initial distribution discretized state matrix across different state sizes on the Image task of LRA benchmark under the Mamba initialization. This task is an Image classification task using the CIFAR-10 dataset (Krizhevsky, 2009), where the inputs are flattened as sequences.

Figure 1 shows distributions of discretized state matrix values across different state sizes in the first layer on the Image task of LRA.

As shown in Figure 1, increasing the state size does not increase the fraction of values close to 1, as analyzed theoretically in the previous subsection. This observation confirms Equation (2). Under the standard Mamba initialization, increasing the state size shifts the discretized state matrix distribution towards values close to 0, without increasing the proportion of values close to 1.

Also, the standard Mamba bias initialization makes the discretized state matrix cover a broad range of time scales. This connects to the literature on bias initialization for gated RNNs, where biases are initialized to provide diverse time constants (Gu et al., 2020). Thus, the Mamba initialization already includes this timescale-aware initialization through the bias. However, in S6, $\bar{A}_{t,i,j} = \exp(-j\Delta_{t,i})$ is not determined by $\Delta_{t,i}$ alone. The multiplication by j shifts the distribution of the discretized state matrix as shown in Figure 1. We therefore aim to reshape the initialization distribution while preserving this broad time-scale coverage as a design constraint.

This analysis motivates us to analyze why values close to 1 in the discretized state matrix are important for preserving previous information over long sequences, which we address in the next section, rather than relying on state-size increases alone.

4 Theoretical analysis of initialization for S6

In this section, we theoretically analyze what the discretized state matrix must satisfy at initialization for S6 to retain information over long sequences. We first consider how long S6 preserves previous information and analyze the impact of input variance.

To isolate the effect of input variability on memory preservation, we analyze the marginal distribution of the input at each time step. At each time step, assume that x_t has marginal distribution $\mathcal{N}(0, \Sigma)$. We do not assume that the inputs are independent across time. Because the input varies from step to step, each subsequent input can change the discretized state matrix, and these changes can erase previously stored information.

We define the per-step expected log-retention rate as

$$\lambda_{i,j} := \mathbb{E}_{x_t}[\log \bar{A}_{t,i,j}]. \tag{3}$$

Because the marginal distribution is the same at each time step, this expectation does not depend on t. By linearity of expectation, the expected cumulative log-retention over T steps is

$$\mathbb{E}\left[\sum_{t=1}^{T} \log \bar{A}_{t,i,j}\right] = T\lambda_{i,j}, \tag{4}$$

which does not require temporal independence. A value of $\lambda_{i,j}$ close to 0 indicates that previous hidden states are preserved over long sequences, while a more negative value indicates faster forgetting. When $\bar{A}_{t,i,j} = 1$ for all t, $\prod_{t=1}^{T} \bar{A}_{t,i,j}$ remains 1, yielding $\lambda_{i,j} = 0$.

From the Mamba initialization in Equation (1), each element of $\mathbf{A} \in \mathbb{R}^{d \times N}$ can be generally written as

$$A_{i,j} = -\alpha_{i,j}, \qquad \alpha_{i,j} > 0. \tag{5}$$

Letting $\mathbf{w}_i^\top$ be the i-th row of $\mathbf{W}_\Delta$ and $\Delta_{t,i} = \mathrm{softplus}(\mathbf{w}_i^\top \mathbf{x}_t + b_{\Delta,i})$, each element of $\bar{\mathbf{A}}_t$ can be written as

$$\bar{A}_{t,i,j} = \exp\big(-\alpha_{i,j}\,\mathrm{softplus}(\mathbf{w}_i^\top \mathbf{x}_t + b_{\Delta,i})\big)\,. \tag{6}$$

Using this expression, we analyze how input variability affects the ability of S6 to preserve previous information over long sequences. We show that the expected log-retention rate $\lambda_{i,j}$ decreases as the projected input variance increases in the following theorem.

Theorem 1 (Projected input variance decreases expected log-retention)**.** *Let $v_i := \mathbf{w}_i^\top \mathbf{\Sigma} \mathbf{w}_i > 0$ be the projected input variance, and $z_{t,i} := \mathbf{w}_i^\top \mathbf{x}_t + b_{\Delta,i}$ be the pre-activation. Since $\mathbf{x}_t \sim \mathcal{N}(\mathbf{0}, \Sigma)$, the pre-activation $z_{t,i}$ satisfies*

$$z_{t,i} = b_{\Delta,i} + \sqrt{v_i}\epsilon, \qquad \epsilon \sim \mathcal{N}(0,1). \tag{7}$$

Thus, the expected log-retention rate $\lambda_{i,j}$ defined in Equation (3) is given as

$$\lambda_{i,j} = -\alpha_{i,j}\mathbb{E}_\epsilon\left[\mathrm{softplus}(b_{\Delta,i} + \sqrt{v_i}\,\epsilon)\right]. \tag{8}$$

Moreover, $\lambda_{i,j}$ strictly decreases as v_i increases.

The proof is provided in Appendix B.1. More generally, the same monotonicity follows from the convexity of softplus. If the projected input becomes more variable while keeping the same mean, the expected step size increases and the expected log-retention decreases. Thus, Theorem 1 formalizes the intuition that input variance can make it difficult to preserve previous information over long sequences. Figure 2 further confirms that the same trend is observed in S6 on the copying task.

When the bias term $b_{\Delta,i}$ is sufficiently negative, it dominates the pre-activation $\mathbf{w}_i^\top \mathbf{x}_t + b_{\Delta,i}$. Consequently, the impact of the input term $\mathbf{w}_i^\top \mathbf{x}_t$ on the pre-activation becomes small. Therefore

$$\mathrm{softplus}(\mathbf{w}_i^\top \mathbf{x}_t + b_{\Delta,i}) \to 0 \tag{9}$$

when $b_{\Delta,i} \to -\infty$, resulting in

$$\bar{A}_{t,i,j} = \exp(-\alpha_{i,j}\Delta_{t,i}) \to 1. \tag{10}$$

Thus we can obtain the discretized state matrix values close to 1 that are less sensitive to the input variance by setting $b_{\Delta,i}$ to a sufficiently negative value.

While our theoretical analysis assumes Gaussian inputs, we conduct a simple experiment to examine whether the same retention mechanism appears in the actual behavior of S6. We use a copying task, which is described in Appendix C.2. In this task, the model is given a sequence of tokens to memorize, followed by dummy tokens that should be ignored, and is evaluated by whether it can reproduce the memorized tokens after the dummy sequence. We vary the bias and the input noise standard deviation at dummy positions.

Results are presented in Figure 2, which shows the training accuracy for dummy sequence lengths of 100 and 200 with 10,000 training steps. For the dummy length of 100, larger input noise standard deviation makes training more difficult, while more negative b_Δ improves the training accuracy. When the dummy length is increased to 200, the task becomes more difficult and requires a more negative b_Δ to achieve high accuracy. Larger input noise standard deviation at dummy positions corresponds to higher projected input variance v_i in Theorem 1.

To interpret the bias dependence, consider the simplified case with $x_t = 0$ and $\alpha_{i,j} = 1$. Then the discretized state matrix is $\bar{A} = \exp(-\mathrm{softplus}(b_\Delta))$. When $b_\Delta \geq -2.0$, we have $\mathrm{softplus}(b_\Delta) > 0.126$, which yields strong contraction; over $L = 100$ steps, $\bar{A}^L$ is nearly 0, explaining the near chance-level accuracy as shown in the columns $b_\Delta \in \{-2.0, -1.0, 0.0\}$ of Figure 2. At $b_\Delta = -3.0$, $\mathrm{softplus}(b_\Delta) \approx 0.049$ and $\bar{A} \approx 0.953$. Thus, $\bar{A}^{100} \approx 0.008$, which is small but nonzero and can

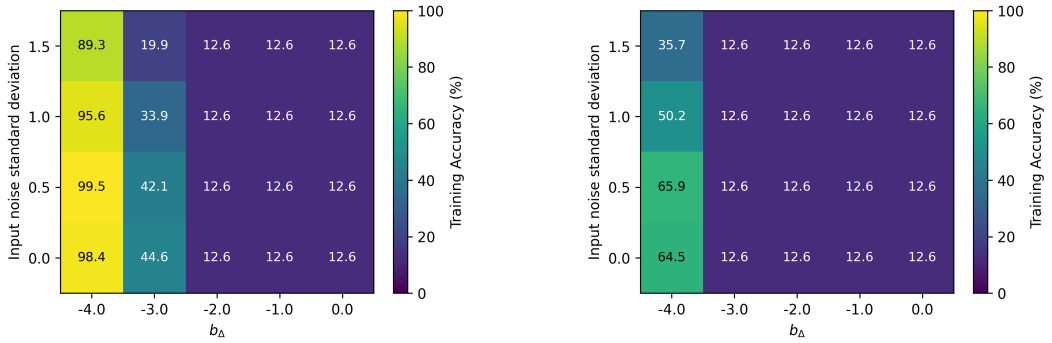


Figure 2: Training accuracy across different b_Δ values and input noise standard deviations on the copying task for the dummy sequence length of 100 (left) and 200 (right).

allow partial retention, consistent with the moderate accuracy for dummy length 100 in the -3.0 column of the left panel. However, $\bar{A}^{200} \approx 6 \times 10^{-5}$, which is effectively zero, corresponds to the near chance-level accuracy for dummy length 200 in the -3.0 column of the right. At $b_\Delta = -4.0$, $\mathrm{softplus}(b_\Delta) \approx 0.018$ and $\bar{A} \approx 0.982$. Then $\bar{A}^{100} \approx 0.163$, providing substantial retention over 100 steps and explaining the high accuracy in the low-noise setting as shown in the -4.0 column of the left. Even for $L = 200$, $\bar{A}^{200} \approx 0.027$, which preserves more signal than the less negative bias settings and is consistent with the observed partial retention in the -4.0 column of the right.

These results are consistent with our analysis that input variability makes it difficult to preserve previous information over long sequences, whereas sufficiently negative b_Δ provides discretized state matrix values close to 1 that help preserve previous states.

When $\bar{A}_{t,i,j}$ is close to 0 or 1, the discretized state matrix barely responds to changes in the input $\mathbf{x}_t$, making it difficult to control the discretized state matrix through learning. To flexibly adjust the discretized state matrix through the input, it is important that $\bar{A}_{t,i,j}$ lies in the middle regime, as shown in the following Theorem.

Theorem 2 (Maximum input sensitivity)**.** *Let $z_{t,i} := \mathbf{w}_i^\top \mathbf{x}_t + b_{\Delta,i}$ be a pre-activation. The sensitivity of $\bar{A}_{t,i,j}$ to the input, measured by the absolute derivative with respect to the pre-activation, is*

$$\left| \frac{\partial \bar{A}_{t,i,j}}{\partial z_{t,i}} \right| = \alpha_{i,j}\, \bar{A}_{t,i,j} \left(1 - \bar{A}_{t,i,j}^{1/\alpha_{i,j}} \right). \tag{11}$$

This sensitivity converges to 0 as $\bar{A}_{t,i,j} \to 0$ or $\bar{A}_{t,i,j} \to 1$, and attains its maximum at an interior point of $(0, 1)$. Moreover, under the Mamba initialization with $\alpha_{i,j} \geq 1$, the value $\bar{A}^*_{t,i,j}$ that maximizes the sensitivity lies in the range $[e^{-1},\ 0.5] \approx [0.368,\ 0.5]$.

The proof is provided in Appendix B.2. Theorem 2 indicates that pushing the discretized state matrix values close to 1 by setting a sufficiently negative $b_{\Delta,i}$ causes their sensitivity to the input to vanish, making it difficult to learn from inputs.

Therefore, combining Theorems 1 and 2, the desirable initialization has the following two types of components: In **Theorem 1**, values of discretized state matrix are close to 1, where $b_{\Delta,i}$ is set sufficiently negative to suppress the impact of input variance. In **Theorem 2**, discretized state matrix $\bar{A}_{t,i,j}$ lies in the middle regime ($[0.368,\ 0.5]$), enabling it to optimize through learning signals from the input.

5 Initialization of Discretized State Matrix Using the Gumbel-Softmax Trick

Theorem 1 and Theorem 2 suggest that the initialized discretized state matrix $\bar{A}_{t,i,j}$ should contain both values close to 1, which preserve previous information over long sequences, and middle-regime values, which can be flexibly adjusted through the inputs. These properties specify a desired shape of the initialization distribution, rather than a unique initialization rule.

Our goal is therefore to construct a simple initialization approach that can reshape the distribution of the bias initialization without changing the S6 architecture.

We use the Gumbel-Softmax trick (Li et al., 2018) for bias initialization because it includes the Mamba initialization as a special case and provides two-parameter control over the distribution shape by changing only the sampling distribution used for bias initialization.

Bias initialization with the Gumbel-Softmax trick. The *Gumbel-Softmax trick* (Maddison et al., 2017; Jang et al., 2017), leveraged by Li et al. (2018), is given as follows:

$$G(\alpha, \tau) = \sigma\left(\frac{\alpha + \log(u_G) - \log(1 - u_G)}{\tau}\right), \tag{12}$$

where $\alpha \in \mathbb{R}$ is a parameter, $\tau > 0$ is a temperature parameter, and $u_G \sim \mathcal{U}[0, 1]$ is sampled from the uniform distribution. We apply this trick to the bias initialization in Equation (1); that is, u is sampled as $u \sim G(\alpha, \tau)$. By changing (α, τ), we can control the distribution of the initialized discretized state matrix, increasing the number of values close to 1 while preserving middle-regime values that can be flexibly adjusted through the inputs.

Gumbel-Softmax trick for bias initialization is a generalization of the standard Mamba initialization, which corresponds to the case when $\alpha = 0.0$ and $\tau = 1.0$ in the Gumbel-Softmax trick. We can see this from the definition since $\sigma^{-1}(x) = \mathrm{logit}(x) = \log\big(x/(1-x)\big)$ and it follows that

$$G(0, 1) = \sigma(\log u - \log(1 - u)) = \sigma\Big(\log\big(\frac{u}{1-u}\big)\Big) = \sigma(\mathrm{logit}(u)) = u. \tag{13}$$

Parameter selection for initialization properties. We tune the two parameters α and τ of the Gumbel-Softmax trick to satisfy the initialization properties derived from Theorems 1 and 2.

Theorem 1 suggests that values of $\bar{A}$ close to 1 are important for very long-term dependencies. We therefore use $P(\bar{A} \geq 0.999)$ to measure the ratio of values that are extremely close to 1. Theorem 2 also suggests that middle-regime values ($[0.368, 0.5]$) are required for input-dependent control. We therefore use $P(0.368 \leq \bar{A} \leq 0.5)$. In addition, the standard Mamba initialization provides a broad range of time scales (Section 3.2). We therefore use $P(\bar{A} \geq 0.8)$ to measure whether this broad coverage is preserved.

The parameter α shifts $G(\alpha, \tau)$ toward 1, making $\Delta_{t,i}$ larger and thus the discretized state matrix $\bar{A}_{t,i,j} = \exp(-\alpha_{i,j}\Delta_{t,i})$ close to 0. On the other hand, decreasing α makes $\bar{A}_{t,i,j}$ close to 1. The parameter τ controls the sharpness. When $\tau \to 0$, the resulting distribution can be considered as an approximation of the Bernoulli distribution (Li et al., 2018), where the distribution is sharp. Moreover, when τ and α are 1.0 and 0.0, respectively, the distribution can be considered as the uniform distribution, which is the standard initialization as discussed in Equation (13).

We obtain Table 1 by Monte Carlo sampling from the initialization distribution. For each (α, τ), we draw 150,000 samples $u_G \sim \mathcal{U}[0, 1]$ and compute

$$\gamma = \sigma\left(\frac{\alpha + \log u_G - \log(1 - u_G)}{\tau}\right), \qquad \Delta = \exp(\gamma(\log \Delta_{\max} - \log \Delta_{\min}) + \log \Delta_{\min}), \tag{14}$$

with $\Delta_{\min} = 10^{-3}$ and $\Delta_{\max} = 10^{-1}$. In this work, we use a state size of 32 for memory efficiency. In this calculation, we focus on the effect of the bias initialization by ignoring the input contribution, i.e., setting $\mathbf{x}_t = \mathbf{0}$, so that Δ is determined only by the initialized bias.

From Table 1, when $(\alpha, \tau) = (0.5, 1.0)$, the ratio of middle-regime values ($[0.368, 0.5]$) is the largest, while the ratio of $\bar{A} \geq 0.999$ is decreased compared to S6. Therefore, these parameters $(\alpha, \tau) = (0.5, 1.0)$ make it easier to flexibly adjust $\bar{A}$ through the input, but reduce the number of values extremely close to 1. In contrast, when $(\alpha, \tau) = (0.5, 0.5)$, the ratio of $\bar{A} \geq 0.999$ is the largest. However, the ratio of $\bar{A} \geq 0.8$ decreases compared to S6. We therefore seek parameters that balance the derived initialization properties. The setting $(\alpha, \tau) = (0.2, 0.8)$ increases both $P(\bar{A} \geq 0.999)$ and $P(0.368 \leq \bar{A} \leq 0.5)$, while keeping $P(\bar{A} \geq 0.8)$ close to that of the standard S6 initialization. This setting therefore supports extremely long-term dependencies and input-dependent control, while maintaining the broad range of time scales.

Table 1: Initial distribution statistics of the discretized state matrix $\bar{A}$ with state size $N = 32$. The probabilities are computed over all sampled state indices.

Initialization	$P(\bar{A} \geq 0.999)$	$P(\bar{A} \geq 0.8)$	$P(0.368 \leq \bar{A} \leq 0.5)$
S6 ($(\alpha, \tau) = (0.0, 1.0)$)	0.000002 (1.0x)	0.614483 (1.0x)	0.059945 (1.0x)
$(\alpha, \tau) = (0.5, 1.0)$	0.000001 (0.5x)	0.509364 (0.83x)	0.079659 (1.33x)
$(\alpha, \tau) = (0.5, 0.5)$	0.000191 (95.5x)	0.465072 (0.76x)	0.077424 (1.29x)
Proposed $(\alpha, \tau) = (0.2, 0.8)$	0.000017 (8.5x)	0.556955 (0.91x)	0.069450 (1.16x)

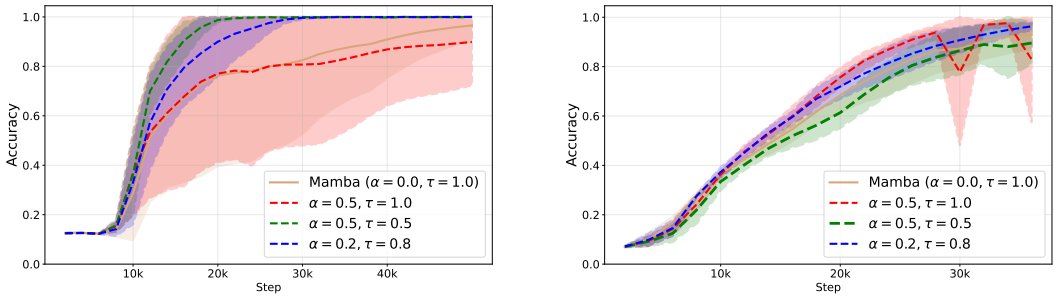


Figure 3: Performances on the copying task (left) and the selective copying task (right).

6 Empirical Verification

We validate our proposed approach on practical settings and datasets. We compare against S6 and validate the effect of initialization. We conduct our experiments in two stages. First, we use synthetic tasks to ablate (α, τ) and validate the theoretical tradeoff. We select $(\alpha, \tau) = (0.2, 0.8)$ as our derived bias initialization. Next, we evaluate this configuration on LRA tasks. In the following experiments, we use 6-layer models, following prior work (Gu et al., 2022a; Gu & Dao, 2024; Yu & Erichson, 2025). We compare our proposed initialization with the original S6 initialization used in Mamba. The implementation details and full experimental setup are provided in Appendix C.

Ablation study on synthetic tasks for the initialization of Gumbel-Softmax. To validate the properties derived from our analysis, we use two synthetic tasks for sequence modeling. The first is the copying task (Arjovsky et al., 2016), which requires preserving the initial tokens through a long sequence of dummy tokens and therefore tests the prediction of Theorem 1. The second is the selective copying task (Jing et al., 2019; Gu & Dao, 2024), a variant of the copying task that requires memorizing tokens at input-dependent positions and therefore tests the prediction of Theorem 2. We use these tasks to ablate the Gumbel-Softmax parameters (α, τ).

Figure 3 (left) shows the mean accuracies and standard deviations across three random seeds on the copying task. The model with $(\alpha, \tau) = (0.2, 0.8)$ converges faster than the S6 baseline since its ratio of values with $\bar{A} \geq 0.999$ is larger as shown in Table 1. This is consistent with Theorem 1. Also, $(\alpha, \tau) = (0.5, 0.5)$ converges fastest because its ratio of values with $\bar{A} \geq 0.999$ is largest among all models, which helps to capture long-term dependencies. In contrast, $(\alpha, \tau) = (0.5, 1.0)$ converges slower than the S6 baseline because its ratio of values with $\bar{A} \geq 0.999$ is smaller, making it difficult to capture long-term dependencies. We further visualize the corresponding discretized state matrix distributions before and after training in Appendix C.9. These results are consistent with Theorem 1.

Figure 3 (right) shows the mean accuracies and standard deviations across three random seeds on the selective copying task. The model with $(\alpha, \tau) = (0.2, 0.8)$ learns faster than the S6 baseline because its discretized state matrix has a large ratio of the middle-regime values ($[0.368, 0.5]$), while keeping values close to 1. In contrast, the model with $(\alpha, \tau) = (0.5, 0.5)$ learns slower than the S6 baseline because its ratio of values with $\bar{A} \geq 0.8$ is smaller, making it difficult to capture long-term dependencies. Finally, $(\alpha, \tau) = (0.5, 1.0)$ learns the fastest among all models, since the model has the largest ratio with the middle-regime values ($[0.368, 0.5]$), but shows larger variance across seeds later in training. This suggests that the large ratio of middle-regime values compared to values close to 1 can make the discretized state matrix too sensitive to inputs. Overall, the model with $(\alpha, \tau) = (0.2, 0.8)$ balances long-term dependencies and input sensitivity.

Table 2: Classification accuracies on ListOps, Text, Retrieval, Image, Pathfinder, and Path-X tasks.

Model	ListOps	Text	Retrieval	Image	Pathfinder	Path-X
S6 ($\alpha = 0.0, \tau = 1.0$)	55.80 ± 3.10	88.06 ± 0.27	91.00 ± 0.12	77.12 ± 2.89	79.56 ± 3.14	73.34 ± 12.01
Proposed ($\alpha = 0.2, \tau = 0.8$)	$\mathbf{57.32 \pm 3.66}$	$\mathbf{88.32 \pm 0.32}$	$\mathbf{91.10 \pm 0.19}$	$\mathbf{81.62 \pm 2.30}$	$\mathbf{83.64 \pm 1.73}$	$\mathbf{83.86 \pm 2.57}$

Table 3: Effect of state size on Image, Pathfinder, and Text. Expanding the state size to $N = 64$ provides smaller gains than the proposed initialization at $N = 32$.

Model	Image	Pathfinder	Text
S6 ($N = 32$, $(\alpha, \tau) = (0.0, 1.0)$)	77.12 ± 2.89	79.56 ± 3.14	88.06 ± 0.27
S6 ($N = 64$, $(\alpha, \tau) = (0.0, 1.0)$)	78.88 ± 3.46	81.56 ± 0.82	87.50 ± 0.67
Proposed ($N = 32$, $(\alpha, \tau) = (0.2, 0.8)$)	$\mathbf{81.62} \pm 2.30$	$\mathbf{83.64} \pm 1.73$	$\mathbf{88.32} \pm 0.32$

These results are consistent with Theorem 1 and Theorem 2, although Theorem 1 is derived under a Gaussian input model.

Evaluation on Long Range Arena. We evaluate whether our initialization improves the preservation of previous information over long sequences. Table 2 shows the mean accuracies and standard deviations across 5 random seeds on the LRA benchmark tasks. Our initialization substantially improves performance on Image, Pathfinder, and Path-X. These improvements over the S6 baseline are consistent with Theorem 1: increasing the ratio of values close to 1 in the discretized state matrix helps preserve information over long sequences. Our initialization also maintains comparable performance on the remaining tasks. In particular, the proposed initialization maintains performance comparable to the S6 baseline on the context-dependent Text task, which is consistent with Theorem 2. By preserving middle-regime values in the discretized state matrix while increasing the proportion of values close to 1, the initialization maintains the input sensitivity that context-dependent tasks require. These results are consistent with our analysis that increasing the ratio of values close to 1 in the discretized state matrix improves the ability to preserve context-independent information over long sequences such as Image and Pathfinder, while maintaining comparable performance on Text, which requires context-dependent memory updates.

Effect of state size. Finally, we compare our initialization with simply increasing the state size of S6. We evaluate on Image and Pathfinder, which require preserving information over long sequences, and on Text, which requires capturing context-dependent information. As shown in Table 3, increasing N from 32 to 64 moderately improves the S6 baseline on Image and Pathfinder, but the proposed initialization with $N = 32$ still outperforms S6 with $N = 64$ on Image and Pathfinder, while maintains comparable performance on Text. This is consistent with Figure 1: under the standard Mamba initialization, increasing the state size mainly adds values close to 0 and does not increase the proportion of values close to 1. These results suggest that reshaping the distribution is at least as important as increasing the state size for preserving information over long sequences.

7 Conclusion

We studied how input variability and initialization affect long-term dependencies in S6. We showed that larger input variance makes it more difficult to preserve information over long sequences, suggesting that the discretized state matrix should contain values close to 1 at initialization. We further showed that if all elements are pushed to 1, the sensitivity of the discretized state matrix to the input vanishes, making it difficult to learn from inputs. We also showed that the standard Mamba initialization causes a distribution shift in the discretized state matrix depending on the state size and increasing the state size shifts the discretized state matrix distribution toward values close to 0, without increasing the proportion of values close to 1. Based on this analysis, we introduced a simple Gumbel-Softmax-based bias initialization that increases values close to 1 while preserving middle-regime values for input-dependent control, without changing the S6 architecture. Experiments on synthetic copying and LRA tasks support the mechanism, improving retention-oriented tasks while maintaining performance on context-dependent tasks. These results suggest that initialization is crucial for long-sequence modeling with selective SSMs.

AI USE STATEMENT

In this work, we used generative AI tools for polishing the text and formatting latex code and figures, modifying experiment code and searching information. We have not used generative AI tools for any important component of the research methodology or design experiments. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary Evolution Recurrent Neural Networks. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

Assaf Ben-Kish, Itamar Zimerman, Shady Abu-Hussein, Nadav Cohen, Amir Globerson, Lior Wolf, and Raja Giryes. DeciMamba: Exploring the length extrapolation potential of Mamba. In *International Conference on Learning Representations*, 2025.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*. PMLR, 2024.

Soham De, Samuel L. Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, Guillaume Desjardins, Arnaud Doucet, David Budden, Yee Whye Teh, Razvan Pascanu, Nando De Freitas, and Caglar Gulcehre. Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models. *arXiv preprint arXiv:2402.19427*, 2024.

Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. In *Conference on Language Modeling*, 2024.

Albert Gu, Caglar Gulcehre, Tom Paine, Matt Hoffman, and Razvan Pascanu. Improving the Gating Mechanism of Recurrent Neural Networks. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Albert Gu, Karan Goel, and Christopher Ré. Efficiently Modeling Long Sequences with Structured State Spaces. In *International Conference on Learning Representations*, 2022a.

Albert Gu, Ankit Gupta, Karan Goel, and Christopher Ré. On the Parameterization and Initialization of Diagonal State Space Models. In *Advances in Neural Information Processing Systems*, 2022b.

Ningyuan Huang, Miguel Sarabia, Abhinav Moudgil, Pau Rodriguez, Luca Zappella, and Federico Danieli. Understanding Input Selectivity in Mamba: Impact on Approximation Power, Memorization, and Associative Recall Capacity. In *Proceedings of The 43rd International Conference on Machine Learning*, 2026.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical Reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*, 2017.

Li Jing, Caglar Gulcehre, John Peurifoy, Yichen Shen, Max Tegmark, Marin Soljacic, and Yoshua Bengio. Gated Orthogonal Recurrent Units: On Learning to Forget. *Neural Computation*, 31(4): 765–783, 2019.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 2020.

Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images, 2009.

Zhuohan Li, Di He, Fei Tian, Wei Chen, Tao Qin, Liwei Wang, and Tie-Yan Liu. Towards Binary-Valued Gates for Robust LSTM Training. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *International Conference on Learning Representations*, 2017.

Eric Martin and Chris Cundy. Parallelizing Linear Recurrent Neural Nets Over Sequence Length. In *International Conference on Learning Representations*, 2018.

Antonio Orvieto, Samuel L. Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting Recurrent Neural Networks for Long Sequences. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Bo Peng, Daniel Goldstein, Quentin Anthony, Alon Albalak, Eric Alcaide, Stella Biderman, Eugene Cheah, Teddy Ferdinan, Haowen Hou, Przemysław Kazienko, Kranthi Kiran GV, Satyapriya Krishna, Ronald McClelland Jr., Niklas Muennighoff, Fares Obeid, Atsushi Saito, Guangyu Song, Haoqin Tu, Ruichong Zhang, Bingchen Zhao, Qihang Zhao, Jian Zhu, and Rui-Jie Zhu. Eagle and Finch: RWKV with Matrix-Valued States and Dynamic Recurrence. In *Conference on Language Modeling*, 2024.

Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Xingjian Du, Haowen Hou, Jiaju Lin, Jiaxing Liu, Janna Lu, William Merrill, Guangyu Song, Kaifeng Tan, Saiteja Utpala, Nathan Wilce, Johan S. Wind, Tianyi Wu, Daniel Wuttke, and Christian Zhou-Zheng. RWKV-7 "Goose" with Expressive Dynamic State Evolution. In *Conference on Language Modeling*, 2025.

Zhen Qin, Songlin Yang, and Yiran Zhong. Hierarchically Gated Recurrent Neural Network for Sequence Modeling. In *Advances in Neural Information Processing Systems*, 2023.

Zhen Qin, Songlin Yang, Weixuan Sun, Xuyang Shen, Dong Li, Weigao Sun, and Yiran Zhong. HGRN2: Gated Linear RNNs with State Expansion. In *Conference on Language Modeling*, 2024.

Dragomir R. Radev, Pradeep Muthukrishnan, Vahed Qazvinian, and Amjad Abu-Jbara. The ACL anthology network corpus. *Language Resources and Evaluation*, 47(4):919–944, 2013.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear Transformers Are Secretly Fast Weight Programmers. In *Proceedings of The 38rd International Conference on Machine Learning*, 2021.

Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks. In *International Conference on Learning Representations*, 2019.

Jimmy T. H. Smith, Andrew Warrington, and Scott W. Linderman. Simplified State Space Layers for Sequence Modeling. In *International Conference on Learning Representations*, 2023.

Corentin Tallec and Yann Ollivier. Can recurrent neural networks warp time? In *International Conference on Learning Representations*, 2018.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long Range Arena: A Benchmark for Efficient Transformers. In *International Conference on Learning Representations*, 2021.

Alexandre Torres-Leguet. mamba.py: A simple, hackable and efficient mamba implementation in pure pytorch and mlx., 2024. URL `https://github.com/alxndrTL/mamba.py`.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated Linear Attention Transformers with Hardware-Efficient Training. In *Proceedings of the 41st International Conference on Machine Learning*, 2024a.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing Linear Transformers with the Delta Rule over Sequence Length. In *Advances in Neural Information Processing Systems*, 2024b.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated Delta Networks: Improving Mamba2 with Delta Rule. In *International Conference on Learning Representations*, 2025.

Annan Yu and N. Benjamin Erichson. Block-Biased Mamba for Long-Range Sequence Processing. In *Advances in Neural Information Processing Systems*, 2025.

A Related work

A.1 Linear recurrent models

Linear recurrent models have achieved competitive performance as an alternative to transformers on modeling long-sequence tasks. Many variants of linear RNNs, including GILR (Martin & Cundy, 2018), S4 (Gu et al., 2022a), S5 (Smith et al., 2023), LRU (Orvieto et al., 2023), Griffin (De et al., 2024), and HGRN (Qin et al., 2023), have proven to be able to capture long-term dependencies. Recently, new types of linear recurrent models (Katharopoulos et al., 2020) emerged based on linear attentions. They address the limitation of the expressiveness of linear RNNs. Several studies have explored gated linear recurrent models with linear attention, such as DeltaNet(Schlag et al., 2021; Yang et al., 2024b), Gated Linear Attention (Yang et al., 2024a), Mamba2 (Dao & Gu, 2024), HGRN2 (Qin et al., 2024), Gated Delta Network (Yang et al., 2025), and RWKV (Peng et al., 2024; 2025), which have demonstrated competitive performance on modeling long-sequence tasks. Unlike these architecture-level modifications, we focus on the standard S6 recurrence and study how its standard initialization shapes the distribution of the discretized state matrix. B2S6 (Yu & Erichson, 2025) and DeciMamba (Ben-Kish et al., 2025) improve S6 by modifying the architecture for long-range tasks. Recent work has shown that input-dependent gates can adapt their values to the context, thereby helping preserve relevant information (Huang et al., 2026). However, Yu & Erichson (2025) show that input-dependent gates can also make the dynamics sensitive to task-irrelevant inputs, particularly under channel-specific input projections, thereby degrading the model's ability to capture long-term dependencies. In contrast, our work studies an issue in the original S6 dynamics and changes only the bias initialization. Thus, our contribution is not a new recurrent architecture, but an analysis of the degradation for long-term dependencies and a simple initialization approach that mitigates it without changing the model architecture.

A.2 Optimization for initializing biases for linear RNNs

Efficient optimization is crucial for RNNs. Standard initialization struggles to optimize parameters because gradients through gates vanish when gate values are close to 0 or 1. In nonlinear RNNs, Tallec & Ollivier (2018) introduced a bias initialization so that gate values become high values to capture long-term dependencies. Li et al. (2018) introduced Gumbel-Gate which pushes gate values towards zero or one by replacing the sigmoid activation function with the Gumbel-Gate. Gu et al. (2020) introduced a combination of the gate mechanism and bias initialization to push gate values close to zero or one. ON-LSTM (Shen et al., 2019) explicitly orders neurons to capture dependencies with different time scales. In linear RNNs, HGRN (Qin et al., 2023) is inspired by ON-LSTM (Shen et al., 2019) and introduces lower bounds for gate values. The lower bound increases monotonically from input to output layers, enabling storing information with different time scales at different layers. The standard Mamba initialization induces a broad range of time scales through its bias initialization. This property is related to prior bias-initialization approaches for gated RNNs, where biases are used to provide a broad range of time scales (Tallec & Ollivier, 2018; Gu et al., 2020).

B Proofs

B.1 Proof of Theorem 1

Proof. Since $\mathbf{x}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$, the pre-activation $z_{t,i} := \mathbf{w}_i^\top \mathbf{x}_t + b_{\Delta,i}$ satisfies

$$z_{t,i} \overset{d}{=} b_{\Delta,i} + \sqrt{v_i}\,\epsilon, \qquad \epsilon \sim \mathcal{N}(0,1).$$

Therefore,

$$\mathbb{E}_{\mathbf{x}_t}[\mathrm{softplus}(z_{t,i})] = \mathbb{E}_\epsilon[\mathrm{softplus}(b_{\Delta,i} + \sqrt{v_i}\,\epsilon)]. \tag{15}$$

Differentiating with respect to v_i, we have

$$\frac{\partial}{\partial v_i}\mathbb{E}_\epsilon[\mathrm{softplus}(b_{\Delta,i} + \sqrt{v_i}\,\epsilon)] = \mathbb{E}_\epsilon\left[\mathrm{softplus}'(b_{\Delta,i} + \sqrt{v_i}\,\epsilon)\frac{\epsilon}{2\sqrt{v_i}}\right] \tag{16}$$

$$= \frac{1}{2\sqrt{v_i}}\mathbb{E}_\epsilon\left[\epsilon\,\mathrm{softplus}'(b_{\Delta,i} + \sqrt{v_i}\,\epsilon)\right], \tag{17}$$

Since $\epsilon \sim \mathcal{N}(0,1)$, from integration by parts we have,

$$\frac{1}{2\sqrt{v_i}}\mathbb{E}_\epsilon\left[\epsilon\,\mathrm{softplus}'(b_{\Delta,i}+\sqrt{v_i}\,\epsilon)\right].=\frac{1}{2}\mathbb{E}_\epsilon\left[\mathrm{softplus}''(b_{\Delta,i}+\sqrt{v_i}\,\epsilon)\right]. \tag{18}$$

Since the derivative of softplus is sigmoid,

$$\mathrm{softplus}''(z)=\sigma(z)(1-\sigma(z))>0 \tag{19}$$

For all z, we have

$$\frac{\partial\lambda_{i,j}}{\partial v_i}=-\frac{\alpha_{i,j}}{2}\mathbb{E}_\epsilon[\mathrm{softplus}''(b_{\Delta,i}+\sqrt{v_i}\,\epsilon)]<0. \tag{20}$$

Therefore, $\lambda_{i,j}$ is strictly decreasing in v_i. □

B.2 Proof of Theorem 2

Proof. From $\Delta_{t,i}=\mathrm{softplus}(z_{t,i})$, by the chain rule,

$$\frac{\partial\bar{A}_{t,i,j}}{\partial z_{t,i}}=\frac{\partial\bar{A}_{t,i,j}}{\partial\Delta_{t,i}}\cdot\frac{\partial\Delta_{t,i}}{\partial z_{t,i}}. \tag{21}$$

Since the derivative of softplus is sigmoid,

$$\frac{\partial\Delta_{t,i}}{\partial z_{t,i}}=\frac{\partial}{\partial z_{t,i}}\log(1+e^{z_{t,i}})=\sigma(z_{t,i}). \tag{22}$$

Therefore,

$$\frac{\partial\bar{A}_{t,i,j}}{\partial z_{t,i}}=(-\alpha_{i,j}\,\bar{A}_{t,i,j})\cdot\sigma(z_{t,i}). \tag{23}$$

From the definition of softplus,

$$\bar{A}_{t,i,j}=\exp(-\alpha_{i,j}\log(1+e^{z_{t,i}}))=(1+e^{z_{t,i}})^{-\alpha_{i,j}}. \tag{24}$$

Also,

$$\sigma(z_{t,i})=\frac{e^{z_{t,i}}}{1+e^{z_{t,i}}}=1-\frac{1}{1+e^{z_{t,i}}}, \tag{25}$$

so

$$\sigma(z_{t,i})=1-\bar{A}_{t,i,j}^{1/\alpha_{i,j}}. \tag{26}$$

Then we can get the sensitivity formula in the Theorem. Since $\bar{A}_{t,i,j}$ is continuous on $(0,1)$ and the sensitivity converges to 0 at both ends $\bar{A}_{t,i,j}\to 0$ and $\bar{A}_{t,i,j}\to 1$, the maximum is attained at an interior point of $(0,1)$.

Next, we compute the value $\bar{A}^*_{t,i,j}$ that maximizes the sensitivity. Since $\bar{A}_{t,i,j}=\exp(-\alpha_{i,j}\Delta_{t,i})$, we have $\bar{A}_{t,i,j}^{1/\alpha_{i,j}}=e^{-\Delta_{t,i}}$, so the sensitivity can be written as

$$\left|\frac{\partial\bar{A}_{t,i,j}}{\partial z_{t,i}}\right|=\alpha_{i,j}\,\exp(-\alpha_{i,j}\Delta_{t,i})(1-e^{-\Delta_{t,i}})=\alpha_{i,j}\left(e^{-\alpha_{i,j}\Delta_{t,i}}-e^{-(\alpha_{i,j}+1)\Delta_{t,i}}\right). \tag{27}$$

Maximizing with respect to $\Delta_{t,i}$, we get the following equation:

$$\frac{\partial}{\partial\Delta_{t,i}}\left|\frac{\partial\bar{A}_{t,i,j}}{\partial z_{t,i}}\right|=\alpha_{i,j}\left(-\alpha_{i,j}e^{-\alpha_{i,j}\Delta_{t,i}}+(\alpha_{i,j}+1)e^{-(\alpha_{i,j}+1)\Delta_{t,i}}\right)=0. \tag{28}$$

Therefore,

$$e^{-\Delta_{t,i}}=\frac{\alpha_{i,j}}{\alpha_{i,j}+1}. \tag{29}$$

That is, the optimal $\Delta^*_{t,i}$ is $\log\left(1+\frac{1}{\alpha_{i,j}}\right)$, and

$$\bar{A}^*_{t,i,j}=\left(1+\frac{1}{\alpha_{i,j}}\right)^{-\alpha_{i,j}}. \tag{30}$$

Then, when $\alpha_{i,j}=1$, $\bar{A}^*_{t,i,j}=0.5$ and when $\alpha_{i,j}\to\infty$, $\bar{A}^*_{t,i,j}\to e^{-1}$. Since $\bar{A}^*_{t,i,j}$ is monotonically decreasing in $\alpha_{i,j}$, it lies in $[e^{-1},\ 0.5]\approx[0.368,\ 0.5]$ for $\alpha_{i,j}\geq 1$. □

C EXPERIMENTS

C.1 IMPLEMENTATIONS

We rely on the code provided by Gu et al. (2022a)[1] for the ListOps, Text, Retrieval, and Image tasks, with Mamba implemented using (Torres-Leguet, 2024). For the Pathfinder and Path-X task, we rely on the code provided by Yu & Erichson (2025).

C.2 EXPERIMENTAL SETUP FOR SYNTHETIC TASKS

Copying task. For the copying task (Arjovsky et al., 2016), the input sequence consists of M memorizing tokens, T dummy tokens, and M output tokens. The model must memorize the first M tokens and reproduce them in the last M steps. From K tokens $(0, 1, \ldots, K-1)$, the first M tokens are randomly sampled from $(1, \ldots, K-2)$, the middle T dummy tokens are set to 0, and the last M output tokens are set to $K-1$. We set $K = 10$, $M = 10$, and $T = 500$, resulting in a total sequence length 520.

Selective copying task. For the selective copying task (Jing et al., 2019; Gu & Dao, 2024), the model must memorize tokens at varying positions instead of the first M tokens. We set $K = 16$, $M = 16$, and $T = 500$, resulting in a total sequence length 532.

For both synthetic tasks, the hidden dimension and embedding dimension are set to 64. We train models with the AdamW optimizer using a learning rate 0.0001, batch size 64, $\Delta_{\max} = 0.1$, and $\Delta_{\min} = 0.001$.

C.3 EXPERIMENTAL SETUP FOR LISTOPS

The ListOps task is a ten-class classification task in which inputs are sequences of up to $2{,}048$ tokens representing hierarchically structured mathematical expressions consisting of nested operations.

Models are trained for 100 epochs with the AdamW optimizer using cosine annealing, peak learning rate 0.004, $30{,}000$ warmup steps, betas $(0.9, 0.999)$, weight decay 0.05, batch size 32, $\Delta_\max = 0.1$, and $\Delta_\min = 0.001$. The hidden dimension is 128 with dropout rate 0.0. Following Yu & Erichson (2025), we set the learning rates of $\mathbf{\Delta}$ and $\mathbf{A}$ to 0.0004 and 0.001, respectively, to improve training stability.

C.4 EXPERIMENTAL SETUP FOR TEXT

The Text task is a classification task using the IMDb dataset, where inputs are represented as sequences of fixed length 4096. Models are trained for 50 epochs with the AdamW optimizer using cosine annealing, peak learning rate 0.005, 3500 warmup steps, betas $(0.9, 0.999)$, weight decay 0.05, batch size 32, $\Delta_{\max} = 0.1$, and $\Delta_{\min} = 0.001$. The hidden dimension is 128 with dropout rate 0.0. The learning rates of $\mathbf{\Delta}$ and $\mathbf{A}$ are set to 0.0004 and 0.001, respectively.

C.5 EXPERIMENTAL SETUP FOR RETRIEVAL

The Retrieval task is a binary classification task based on the ACL Anthology Network (AAN; Radev et al., 2013) dataset, where each input consists of a pair of documents represented as character-level sequences of up to $4{,}000$ tokens each.

Models are trained for 20 epochs with the AdamW optimizer using cosine annealing, peak learning rate 0.004, $4{,}902$ warmup steps, betas $(0.9, 0.999)$, weight decay 0.03, batch size 60, $\Delta_\max = 0.1$, and $\Delta_\min = 0.001$. The hidden dimension is 128 with dropout rate 0.0. The learning rates of $\mathbf{\Delta}$ and $\mathbf{A}$ are set to 0.0004 and 0.001, respectively.

[1] https://github.com/state-spaces/s4

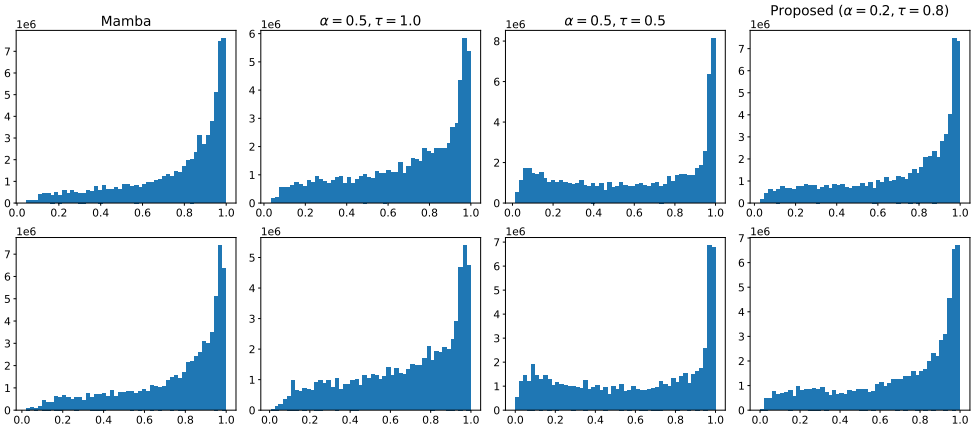


Figure 4: The discretized state matrix distributions before (top) and after (bottom) training under different α and τ for the copying task.

C.6 Experimental setup for Image

The Image task is one of the LRA tasks, where flattened CIFAR-10 images are processed as sequences. Models are trained for 200 epochs with the AdamW optimizer using cosine annealing, peak learning rate 0.004, $18{,}000$ warmup steps, betas $(0.9, 0.999)$, weight decay 0.08, batch size 50, $\Delta_{\max} = 0.1$, and $\Delta_{\min} = 0.001$. The hidden dimension is 512 with dropout rate 0.0. The learning rates of $\mathbf{\Delta}$ and $\mathbf{A}$ are set to 0.0004 and 0.001, respectively.

C.7 Experimental setup for Pathfinder

Pathfinder is one of the LRA tasks, where inputs consist of 32×32 black-and-white images containing randomly drawn lines and two dots. This task requires predicting whether two dots are connected by a line. Images are flattened into sequences of 1024 timesteps. Models are trained for 100 epochs with the AdamW optimizer using cosine annealing, peak learning rate 0.0013, $50{,}000$ warmup steps, betas $(0.9, 0.999)$, weight decay 0.05, batch size 48, $\Delta_{\max} = 0.1$, and $\Delta_{\min} = 0.001$. The hidden dimension is 256 with dropout rate 0.0. The learning rates of $\mathbf{\Delta}$ and $\mathbf{A}$ are set to 0.0004 and 0.001, respectively.

C.8 Experimental setup for Path-X

Path-X is one of the LRA tasks, where each input consists of a 128×128 grayscale image containing two marked endpoints and multiple paths and distractors. The task is to predict whether the two endpoints are connected by a path. Each image is flattened into a sequence of $16{,}384$ timesteps. Models are trained for 50 epochs with the AdamW optimizer using cosine annealing, peak learning rate 0.001, $30{,}000$ warmup steps, betas $(0.9, 0.999)$, weight decay 0.01, batch size 48, $\Delta_{\max} = 0.1$, and $\Delta_{\min} = 0.0001$. The hidden dimension is 128 with dropout rate 0.0. The learning rates of $\mathbf{\Delta}$ and $\mathbf{A}$ are set to 0.00004 and 0.0001, respectively.

C.9 Distribution before and after training

Figure 4 compares the distributional shapes at initialization (top) and after training on the copying task (bottom) for the Mamba baseline and three (α, τ) settings. The figure shows that the overall distributional shapes are preserved after training.