StateX: Enhancing RNN Recall via Post-training State Expansion

Anonymous ACL submission

Abstract

Recurrent neural networks (RNNs), such as linear attention and state-space models, have gained popularity due to their constant per-token complexity when processing long contexts. However, these recurrent models struggle with tasks that require accurate recall of contextual information from long contexts, because all contextual information is compressed into a fixed-size recurrent state. Previous studies have shown that recall ability is positively correlated with the recurrent state size, yet directly training RNNs with large recurrent states results in high training costs. In this paper, we introduce StateX, a post-training framework that efficiently expands the states of pre-trained RNNs. For two popular classes of RNNs, linear attention and state-space models, we design post-training architectural modifications in StateX, to scale up the state size with no or negligible increase in model parameters. Experiments on models with up to 1.3B parameters demonstrate that StateX efficiently enhances the recall and in-context learning performance of RNNs without incurring high post-training costs or compromising other capabilities.

1 Introduction

Recently, recurrent neural networks (RNNs), such as gated linear attention (GLA) (Yang et al., 2024) and Mamba2 (Dao and Gu, 2024), have shown promising performance in building large language models (LLMs). These recurrent architectures have constant per-token complexity for processing long contexts, whereas the widely-used Transformer architecture (Vaswani et al., 2023) has per-token complexity that grows linearly with the context length. Thus, RNNs are much more efficient than Transformers in processing long contexts.

However, RNNs still underperform Transformers in certain aspects, with one of the most critical being the long-context recall ability (Jelassi et al., 2024a). Different from Transformers, which store the representations of all contextual tokens, RNNs compress all contextual information into a fixed-size *state*[1]. As a result, the recall ability of RNNs heavily depends on the capacity of this state (Jelassi et al., 2024b; Arora et al., 2024a; Yang et al., 2025; Chen et al., 2025). Despite the positive gains of increasing the recurrent state size, considering the increased training costs and the limited benefits in short-context scenarios and various downstream tasks, most RNNs are still trained with a small state size relative to the rest of the model. For instance, in Mamba2-1.3B and GLA-1.3B, their recurrent states are smaller than 2% of their model sizes (details in Table 2).

In this paper, we propose StateX, which expands the recurrent state size while keeping training costs low and introducing almost no additional parameters. Specifically, we expand the state size of pre-trained RNNs through post-training on much less data than pre-training. Since larger recurrent states are more important for long-context models, we perform state expansion prior to continual training. The whole pipeline is illustrated in Figure 1.

The state expansion process is an architectural change and depends on the pre-trained model architecture. Therefore, for StateX, we design two state-expansion methods, targeting two popular RNN classes: linear attention models (Katharopoulos et al., 2020; Yang et al., 2024) and state-space models (Dao and Gu, 2024). Additionally, we explore various parameter initialization techniques and select key layers for expansion rather than all layers, to balance model performance and adaptation efficiency. Compared to other state expansion methods that require training from scratch (e.g., MoM (Du et al., 2025)), our method is simpler and can be seamlessly applied to existing effective RNN implementations and training pipelines.

[1]Also named *recurrent state* or *hidden state* in various contexts. Our paper uses these two terms interchangeably.

Method	Performance	Throughput	Training Cost
Vanilla RNNs (small states)	✗ Poor	✓ High	✓ Low
Training large states from scratch	✓	✗ Low	✗ High
Novel architectures with large states (e.g., MoM)	✓	✗ Low	✗ High
StateX (ours)	✓ Good	✓ High	✓ Low

Table 1: Comparison between our work and existing approaches for increasing RNN state sizes. Vanilla RNNs underperform due to their smaller state sizes. "✓" denotes methods that must be trained from scratch at a huge cost. Hence, it is difficult to set up a strictly fair comparison despite their competitive performance.

We evaluate our method on public 1.3B parameter checkpoints of GLA[2] and Mamba2[3], two widely used RNN models, by conducting post-training on 10B tokens. Our empirical results demonstrate that, compared to the traditional two-stage method, StateX significantly improves performance on recall-intensive tasks, in-context learning tasks, and needle-in-a-haystack (NIAH) (Hsieh et al., 2024) tasks while maintaining performance on common-sense reasoning tasks. While using the same amount of training data as ordinary continual training, StateX consistently yields better results: the relative accuracy gains in recall-intensive tasks are 3.36% for GLA and 1.1% for Mamba2, and the relative performance gains in in-context learning are 7.2% for GLA and 1.0% for Mamba2. Also, the average NIAH accuracies (up to 64K context length) improve from 26.0% to 42.2% for GLA, and from 33.2% to 39.2% for Mamba2.

Overall, our contributions include:

- To the best of our knowledge, StateX represents the first work that focuses on expanding the state size of RNNs through post-training.
- For two popular RNN variants, GLA and Mamba2, we design simple and effective state expansion techniques and training recipes for efficient post-training.
- We evaluate our method on public 1.3B checkpoints. Our results show consistent improvements in recall-intensive and in-context learning tasks, without sacrificing performance on common-sense reasoning benchmarks.

2 Related Works

In this section, we provide a brief description of RNNs and related work on expanding their state sizes. For more details about RNNs, please refer to the surveys (Wang et al., 2025; Lv et al., 2025).

[2] https://huggingface.co/fla-hub/gla-1.3B-100B
[3] https://huggingface.co/AntonV/mamba2-1.3b-hf

Modern RNNs Recently, some RNN variants have shown promising results in sequence modeling. Some representative examples include state space models (SSMs) (Dao and Gu, 2024; Gu and Dao, 2024), the RWKV series (Peng et al., 2025, 2024, 2023), linear attention (LA) models (Katharopoulos et al., 2020; Sun et al., 2023; Yang et al., 2024), and DeltaNet (Yang et al., 2025). Some results have shown that these RNNs can outperform Transformers up to several billion parameters on certain language tasks, such as common-sense reasoning (Waleffe et al., 2024; Team, 2024), and some hybrid models have scaled up to over 100B parameters and trillions of training tokens (MiniMax et al., 2025; Team et al., 2025). RNNs are attractive alternatives to Transformers because their per-token complexity is constant, while Transformers' per-token complexity scales linearly with the context length. However, since Transformers cache all contextual token representations, they outperform RNNs in recalling contextual information. This is one of the reasons why RNNs have seen limited adoption.

Increasing RNN State Size Many previous works have investigated the influence of recurrent state size on the capabilities of RNNs. One important improvement of modern RNNs over previous works such as LSTM (Hochreiter and Schmidhuber, 1997) and GRU (Cho et al., 2014) is the adoption of larger matrix-valued recurrent states over smaller vector-valued states (Sun et al., 2023; Qin et al., 2024; Katharopoulos et al., 2020; Hua et al., 2022). Some later efforts focus on improving the forget mechanisms to remove unneeded information in the recurrent states, saving capacity to store more contextual information (Gu and Dao, 2024; Schlag et al., 2021). Arora et al. (2024a) provide a comprehensive comparison of the recall-throughput tradeoff of various recent RNN architectures. Although these methods show promising results, their state size is still rather small, and they lag behind Transformers in recall-intensive tasks.

Recent State Expansion Works More recently, Du et al. (2025) propose MoM, a new architecture that maintains a large state size but with lower computational overhead, by updating only parts of the recurrent state at each time step.

Pan et al. (2025) is a concurrent work to ours that proposes a new row-sparse update formulation for LA, based on which they develop an architecture capable of handling large sparse states, to address the performance degradation in in-context retrieval. Another relevant concurrent work is by Liu et al. (2025). They utilize low-rank projections to increase the state size of RNNs with small parameter overhead, resulting in considerably better recall performance. However, these architectures have not been thoroughly evaluated across different tasks and may be hard to adopt into existing codebases.

In brief, the state size is still a critical bottleneck of RNNs. Increasing the state size provides consistent performance gains for many RNN variants. However, **previous works on expanding RNN states are trained from scratch**, which is highly expensive and requires significant changes to the model architecture and implementation. This paper, to the best of our knowledge, is the first effort to **expand states through post-training**, without training from scratch. Compared to existing architectures with larger states, our method is simpler and can be seamlessly integrated into popular RNN variants such as LA and SSMs. Table 1 shows the comparison between our work and existing works with larger states.

3 Preliminaries

In this section, we first provide a formulation of RNNs as well as two variants—LA and SSM (Sections 3.1, 3.2, and 3.3). Then, we discuss how the recurrent state size influences the models' recall abilities and cost-efficiency (Section 3.4). Since we only modify the RNN block, we omit the formulation for FFN blocks.

3.1 Recurrent Neural Networks

In RNNs, all contextual information is stored in a fixed-size *recurrent state* $\mathbf{S}_t$, where t denotes the time step. At each time step, new information is inserted into the previous state $\mathbf{S}_{t-1}$ with an *update rule* f_{update}, and then retrieves information from $\mathbf{S}_t$ with a *query rule* f_{query}, which is given as

$$\begin{aligned} \mathbf{S}_t &= f_{\text{update}}(\mathbf{S}_{t-1}, \mathbf{x}_t), \\ \mathbf{y}_t &= f_{\text{query}}(\mathbf{S}_t, \mathbf{x}_t), \end{aligned} \tag{1}$$

where $\mathbf{x}_t, \mathbf{y}_t \in \mathbb{R}^d$ are the input and output representations at the time step t. In this paper, we define the *state size* as the parameter count of $\mathbf{S}_t$.

3.2 Linear Attention

There are many LA variants, and we use GLA as a representative example in this study, but our method is applicable to most LA models since they have highly similar formulations. Each GLA layer consists of H heads computed in parallel, and the layer output is the sum of the heads' outputs. Each GLA head can be formulated as:[4]

$$\begin{aligned} \square_{t,h} &= f_{\square,h}(\mathbf{x}_t), \quad \square \in \{\mathbf{q}, \mathbf{k}, \mathbf{v}, \boldsymbol{\alpha}\}, \\ \mathbf{F}_{t,h} &= \text{diag}(\boldsymbol{\alpha}_{t,h}) \\ \mathbf{S}_{t,h} &= \mathbf{F}_{t,h}\mathbf{S}_{t-1,h} + \mathbf{k}_{t,h}^\top \mathbf{v}_{t,h} \in \mathbb{R}^{d_k \times d_v}, \\ \mathbf{y}_{t,h} &= \mathbf{q}_{t,h}\mathbf{S}_{t,h} \in \mathbb{R}^{d_v}, \end{aligned} \tag{2}$$

where $h \in \{1, \cdots, H\}$ is the head index, d_k, d_v are the key and value dimensions, and $f_{\square,h}$ are differentiable functions of $\mathbf{x}_t$. The diagonal structure of the transition matrix $\mathbf{F}_{t,h}$ allows this recurrent form to be parallelized efficiently on modern GPUs (Yang et al., 2024). The state size in each GLA layer is Hd_kd_v.

3.3 State Space Models

We focus on Mamba2, which is a state-of-the-art SSM. A Mamba2 layer can be formulated as:[5]

$$\begin{aligned} \Delta_t &= f_\Delta(\mathbf{x}_t\mathbf{W}_{\Delta,h}) \in \mathbb{R}, \\ \alpha_{t,h} &= \exp(-\Delta_t \exp(A_h)) \in \mathbb{R}, \\ \mathbf{q}_t &= \sigma_q(\mathbf{x}_t\mathbf{W}_q) \in \mathbb{R}^{d_k}, \\ \mathbf{k}_t &= \sigma_k(\mathbf{x}_t\mathbf{W}_k) \in \mathbb{R}^{d_k}, \\ \mathbf{v}_{t,h} &= \sigma_v(\mathbf{x}_t\mathbf{W}_{v,h}) \in \mathbb{R}^{d_v}, \\ \mathbf{S}_{t,h} &= \mathbf{S}_{t-1,h}\alpha_{t,h} + \Delta_{t,h}\mathbf{k}_t^\top \mathbf{v}_{t,h} \in \mathbb{R}^{d_k \times d_v}, \\ \mathbf{y}_{t,h} &= \mathbf{q}_t\mathbf{S}_{t,h} \in \mathbb{R}^{d_v}, \end{aligned} \tag{3}$$

where $\sigma_v, \sigma_k, \sigma_q, f_\Delta$ are differentiable functions, $\mathbf{W}_v \in \mathbb{R}^{d \times d_v}, \mathbf{W}_k, \mathbf{W}_q \in \mathbb{R}^{d \times d_k}, \mathbf{W}_{\Delta,h} \in \mathbb{R}^{d \times 1}, A_h \in \mathbb{R}$ are learnable parameters. d_k and d_v are hyperparameters and are called the *state dimension* and *head dimension* in the SSM literature. The state size of Mamba2 is also Hd_kd_v, although these hyperparameter values may differ from GLA.

[4] We omit some components (e.g., skip connections, normalization) for simplicity.

[5] We use attention notations $(\mathbf{q}_t, \mathbf{k}_t, \mathbf{v}_t)$ instead of SSM notations (x_t, B_t, C_t) from the Mamba2 paper for simplicity and to emphasize the analogy between the two RNN variants.

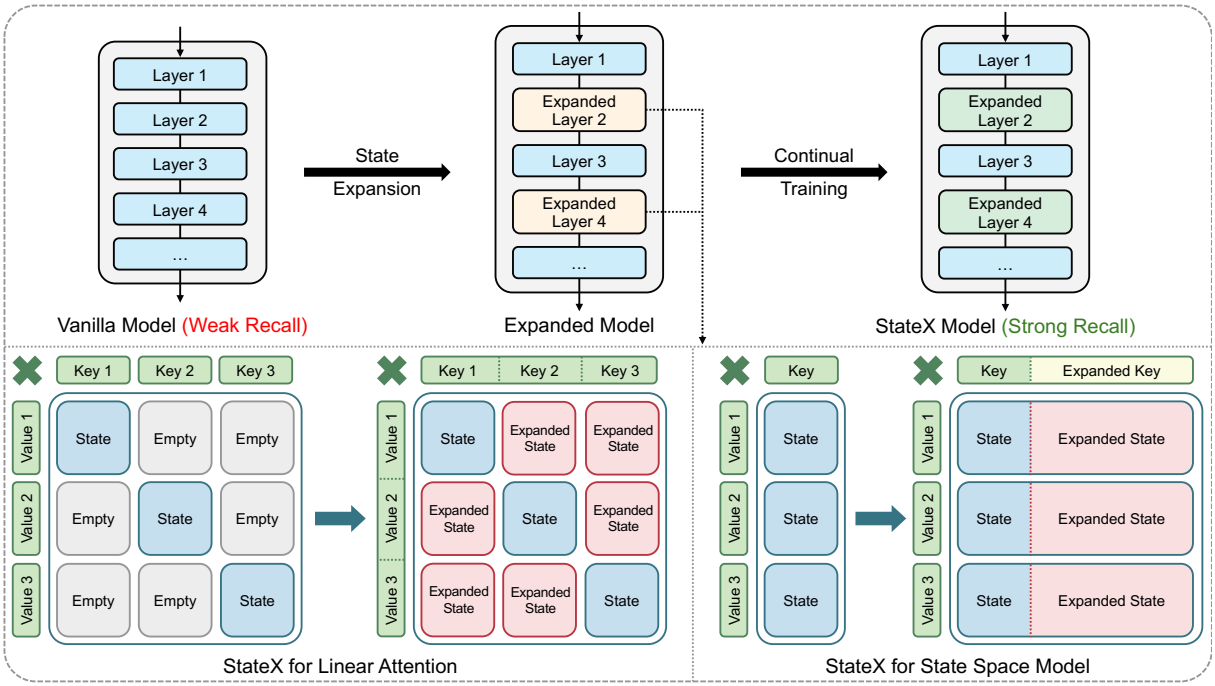

Figure 1: Illustration of the pipeline of StateX and how StateX expands the state size of two RNN variants (linear attention and state space models) with minimal increase in model parameters. The red parts ("Expanded State") indicate the additional state parameters unlocked by StateX.

We provide the complete formulations of Mamba2 in Appendix A.2.

It has been identified that Mamba2 can be viewed as a variant of GLA (Yang et al., 2024), where heads share the same query/key (QK) states. In this paper, we view these two variants as different variants, since this QK sharing influences state expansion, and design expansion methods specifically for GLA and Mamba2, respectively.

3.4 Influence of State Size

Recall Ability Since all contextual information is stored in $\mathbf{S}_t$, the ability of RNNs to recall contextual information depends on the capacity of $\mathbf{S}_t$, which in turn depends on the size of $\mathbf{S}_t$. Extensive empirical evidence indicates a strong positive correlation between the size of the recurrent states and their performance on recall-intensive tasks (Arora et al., 2024a; Hua et al., 2022; Zhang et al., 2025; Jelassi et al., 2024a). These findings highlight the critical role of the state size in determining RNN recall performance, underscoring the importance of state expansion for improving recall abilities.

Efficiency The computational complexity of the token mixing component (i.e., update rule and query rule) scales linearly with the state size. Therefore, blindly increasing the state size can lead to high training and inference costs. StateX alleviates these problems during both training and inference by expanding the states via post-training (so the model is trained with smaller states most of the time) and expanding only a subset of layers.

4 Method

Our method, StateX, involves architectural modifications that expand the RNN state sizes prior to long-context post-training to boost their recall abilities. Meanwhile, we aim to minimize the additional parameters introduced by this modification and keep the final architecture similar to the original architecture to make it easier for the modified model to adapt. An overview of the architectural modifications is illustrated in Figure 1.

In this section, we describe the state expansion recipe for two popular classes of RNNs—GLA (Yang et al., 2024) and SSM (Dao and Gu, 2024) (Sections 4.1 and 4.2). Then, we describe parameter initialization methods after the expansion (Section 4.3) and which layers to expand (Section 4.4).

4.1 StateX for Linear Attention

Since GLA employs a multi-head mechanism with different query, key, and value (QKV) vectors for each head, we can increase the state size by simply merging multiple heads into one larger head. This

is because the state size of H heads is $H \times d_k \times d_v$, and merging them into one head results in a state size of $1 \times Hd_k \times Hd_v$, which is H times larger. Meanwhile, no additional parameters are introduced since the total number of channels in the QKV vectors remains the same. The effect of this change is illustrated in the left side of Figure 1. Merging GLA heads activates non-diagonal regions of the state matrix, thereby achieving larger states than the multi-head counterparts. StateX for GLA can be formulated as:

$$\begin{aligned}
\mathbf{q}'_t &= \begin{bmatrix}\mathbf{q}_{t,1} & \cdots & \mathbf{q}_{t,H}\end{bmatrix} \in \mathbb{R}^{Hd_k}, \\
\mathbf{k}'_t &= \begin{bmatrix}\mathbf{k}_{t,1} & \cdots & \mathbf{k}_{t,H}\end{bmatrix} \in \mathbb{R}^{Hd_k}, \\
\mathbf{v}'_t &= \begin{bmatrix}\mathbf{v}_{t,1} & \cdots & \mathbf{v}_{t,H}\end{bmatrix} \in \mathbb{R}^{Hd_v}, \\
\boldsymbol{\alpha}'_t &= \begin{bmatrix}\boldsymbol{\alpha}_{t,1} & \cdots & \boldsymbol{\alpha}_{t,H}\end{bmatrix} \in \mathbb{R}^{Hd_k}, \\
\mathbf{F}'_t &= \mathrm{diag}(\boldsymbol{\alpha}'_t) \in \mathbb{R}^{Hd_k}, \\
\mathbf{S}'_t &= \mathbf{F}'_t \mathbf{S}'_{t-1} + \mathbf{k}'^{\top}_t \mathbf{v}'_t \in \mathbb{R}^{Hd_k \times Hd_v}, \\
\mathbf{y}_t &= \mathbf{q}'_t \mathbf{S}'_t \in \mathbb{R}^{Hd_v}.
\end{aligned} \tag{4}$$

In practice, we can efficiently implement StateX GLA by updating the values of H, d_k, d_v to $\hat{H} = 1, \hat{d}_k = Hd_k, \hat{d}_v = Hd_v$. Thus, this modification can be seamlessly applied to existing GLA implementations. StateX always merges all heads into one large head, which is motivated by the finding that single-head GLA consistently outperforms multi-head GLA (reported in Section 5.7).

4.2 StateX for SSM

The head merging method is not applicable to SSMs because there is only one query and key vector in each layer. For this RNN variant, we increase the key dimension by expanding the key and query projection layers. Specifically, we increase the hyperparameter d_k (the original Mamba2 paper refers to this as the *state dimension*) by appending new parameters $\left(\hat{\mathbf{W}}_q, \hat{\mathbf{W}}_k \in \mathbb{R}^{d \times (\hat{d}_k - d_k)}\right)$ to $\mathbf{W}_q, \mathbf{W}_k$, where $\hat{d}_k$ is the new expanded key dimension in StateX. StateX for SSM can be formulated as:

$$\begin{aligned}
\mathbf{W}'_q &= \begin{bmatrix}\mathbf{W}_q & \hat{\mathbf{W}}_q\end{bmatrix} \in \mathbb{R}^{d \times \hat{d}_k}, \\
\mathbf{W}'_k &= \begin{bmatrix}\mathbf{W}_k & \hat{\mathbf{W}}_k\end{bmatrix} \in \mathbb{R}^{d \times \hat{d}_k}, \\
\mathbf{q}'_t &= \sigma_q(\mathbf{x}_t \mathbf{W}'_q) \in \mathbb{R}^{\hat{d}_k}, \\
\mathbf{k}'_t &= \Delta_{t,h} \sigma_k(\mathbf{x}_t \mathbf{W}'_k) \in \mathbb{R}^{\hat{d}_k}, \\
\mathbf{S}'_{t,h} &= \mathbf{S}'_{t-1,h} \alpha_{t,h} + \Delta_{t,h} \mathbf{k}'^{\top}_t \mathbf{v}_{t,h} \in \mathbb{R}^{\hat{d}_k \times d_v}, \\
\mathbf{y}_{t,h} &= \mathbf{q}'_t \mathbf{S}'_{t,h} \in \mathbb{R}^{Hd_v}.
\end{aligned} \tag{5}$$

Model	Model size	State size
Linear Attention—GLA		
Vanilla GLA	1.37B	12.6M
StateX (ours)	1.37B	18.9M
State Space Model—Mamba2		
Vanilla Mamba2	1.34B	25.0M
StateX (ours)	1.35B	37.4M
RNN with Sparse State		
MoM	1.55B	31.5M

Table 2: The model and state sizes of GLA and Mamba2 variants as well as MoM.

Since $\mathbf{W}'_k, \mathbf{W}'_q$ are much smaller than the other components, the increase in total parameters is less than 1% when we use $\hat{d}_k = 4d_k$. This modification is illustrated by Figure 1 (right). A complete formulation is given in Appendix A.4.

4.3 Parameter Initialization

After the modification, we can inherit the parameters from the pre-trained model and initialize only the added parameters (for SSMs). However, perhaps surprisingly, we find that inheriting pre-trained parameters can be detrimental to downstream performance. Thus, we present a better parameter initialization strategy. An ablation study on initialization strategies is provided in Section 5.5.

GLA Initialization GLA models consist of interleaving layers of GLA blocks and FFN blocks. After state expansion, we reinitialize all parameters in the GLA blocks, while FFN blocks and the embedding table inherit the pre-trained parameters.

SSM Initialization Mamba2 merges FFN blocks and the SSM mechanism into one unified layer. Motivated by the SSM literature, we only reinitialize the parameters of the SSM mechanism, which are $\mathbf{W}_q, \mathbf{W}_k$, while other modules inherit the pre-trained parameters. Further implementation details can be found in Appendix A.4.

4.4 How Many Layers to Expand?

Expanding the states of all layers may result in a too disruptive change, making it harder for the modified model to recover from this change through post-training with limited data. It may also result in lower inference throughput due to the increase in total state size. Existing works have shown that not all layers are responsible for recalling information (Bick et al., 2025). Thus, we hypothesize that

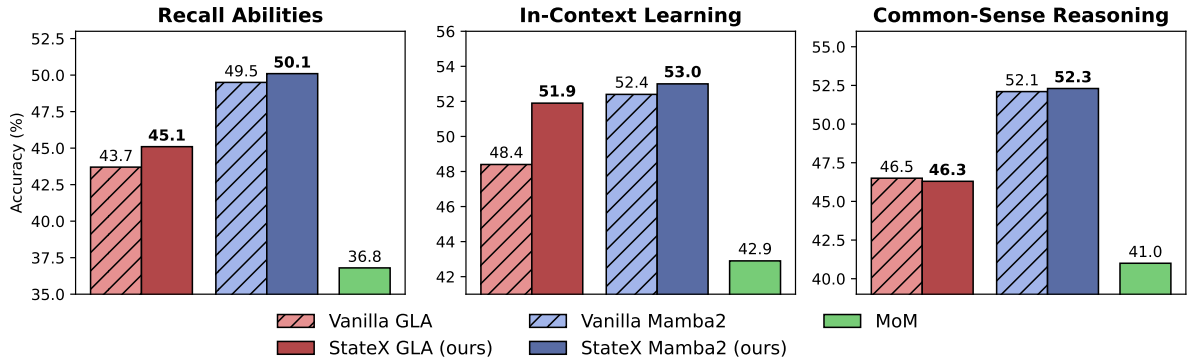

Figure 2: Comparison of the performance of vanilla RNNs (after continual training), StateX (ours), and MoM (RNN with sparse states). StateX improves context utilization abilities without sacrificing common-sense reasoning.

only a subset of layers can benefit from a larger state. Concretely, we adopt a uniform expansion strategy by expanding one layer every m consecutive layers, starting from the first layer. For both GLA and Mamba2, we set the value of m to ensure that the number of expanded layers is exactly 4. In Section 5.6, we empirically ablate the influence of the number of expanded layers. Table 2 reports the model and state sizes of each model studied in this paper. Let N_S denote the state size of vanilla GLA/Mamba2, and $\hat{N}_S$ denote the state size of their StateX versions, then the total state size of StateX is $N_S\frac{L}{m} + \hat{N}_S(1 - \frac{L}{m})$.

5 Experiments

We first describe the details of the experiments (Section 5.1). Then, we present the main results of our method (Section 5.2). We also measure the inference and training efficiency of the models (Sections 5.3 and 5.4). Finally, we provide ablation studies involving the choices of parameter initialization (Section 5.5), the number of expanded layers (Section 5.6), multi-head mechanism in GLA (Section 5.7). Some additional results are reported in Section B.2 and B.4 to save space.

5.1 Experimental Details

Models We apply StateX to the official 1.3B checkpoints of GLA and Mamba2. For Mamba2, StateX increases the d_k hyperparameter from 128 to 512. For GLA, we merge the four existing heads from the pre-trained checkpoint into a single large head. Hence, the expanded layers in the StateX versions have $4\times$ larger states in both models.

Training Data All models are trained on SlimPajama (Soboleva et al., 2023), a widely-used, high-quality, and deduplicated corpus with 627B tokens extracted from the Internet. We concatenate documents with a special token as the delimiter. Then, these concatenations are split into chunks of the specified training context length.

Training Configuration The training follows common practices in context length extension by post-training as closely as possible. Concretely, we use the cosine learning rate scheduler, with a maximum learning rate of 3e-4, and a warmup phase of 5% of the total training steps. We use 64K context length because Gao et al. (2025) have shown that training with longer contexts improves recall capabilities. All models are trained with 10B tokens, with a batch size of 0.5M tokens.

Evaluation We evaluate the models' context utilization abilities with recall-intensive tasks and in-context learning (ICL). The recall-intensive tasks involve 5 popular document question-answering tasks. To assess ICL, we adopt a suite of 7 classification and 5 multiple-choice tasks selected from Min et al. (2022), a study that systematically evaluates ICL capabilities. The ICL prompt contains 16 demonstrations, and the performance is summarized by the mean accuracy averaged over all tasks. Furthermore, we measure the general language processing abilities with 6 popular multiple-choice common-sense reasoning tasks. More details are given in Appendix C.2.

Baselines We mainly compare StateX against vanilla RNNs that have undergone the continual training. The vanilla models undergo the same post-training process, but without any architectural modifications, so their state sizes remain unchanged. We also compare against MoM (Du et al., 2025), which is an RNN with large states with sparse up-

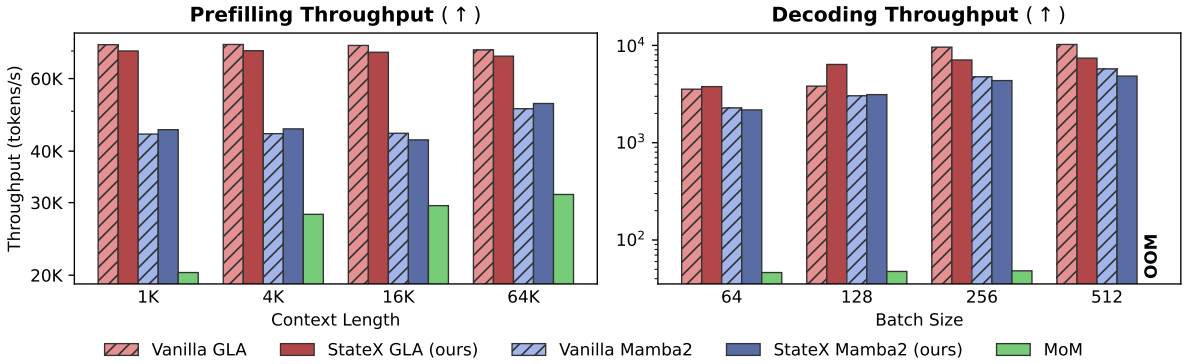

Figure 3: The inference throughput of StateX GLA and Mamba2 versus MoM. StateX improves the RNNs' recall while enjoying great inference speed whereas MoM is much slower in inference. OOM = Out of CUDA memory.

Model	Throughput (tokens/s)
Vanilla GLA	129.1K
StateX GLA	122.1K
Vanilla Mamba	108.5K
StateX Mamba	104.3K
MoM	55.9K

Table 3: Training throughput of vanilla GLA and Mamba2, their StateX versions and MoM. The StateX models have a close throughput to vanilla ones, while they are roughly $2\times$ faster than MoM.

dates[6]. However, as MoM is unable to leverage pre-trained checkpoints, it is trained from scratch. For a fair comparison, throughout the entire continual training process, StateX and each baseline are trained on the same data with the same configuration (details in Appendix C.1).

5.2 Main Results

Figure 2 presents the scores of each model on recall-intensive, in-context learning, and common-sense reasoning tasks. The scores on each task within these three domains are given in Appendix B.1.

Takeaways StateX considerably improves the recall and in-context learning abilities of GLA and Mamba2 over the vanilla versions. StateX also outperforms MoM by a large margin since it can benefit from inheriting abilities from the pre-trained checkpoint while MoM has to learn from scratch.

5.3 Inference Efficiency Results

Here, we measure the inference throughput of the vanilla GLA and Mamba2, their StateX versions, and MoM (Du et al., 2025). The RNN mixer of each model is implemented with kernels from the widely-used Flash Linear Attention GitHub repository[7]. Inference throughput measurements are performed on a NVIDIA A800-SXM4-80GB GPU. The concrete throughput values and additional details are reported in Section B.3.

Takeaways StateX versions of GLA and Mamba2 almost as fast as the original models in prefilling, and slightly slower in decoding for larger batch sizes. Compared to MoM, StateX is $1.7\times$ to $2.5\times$ faster during prefilling and $81\times$ to $147\times$ faster during decoding.

5.4 Training Efficiency Results

Table 3 reports the training efficiency of vanilla RNNs, StateX versions of them, and MoM. Training throughput is measured on a machine equipped with eight NVIDIA A800-SXM4-80GB GPUs. The training framework is implemented with the popular HuggingFace Accelerate framework with data parallelism, a common approach for single-machine, multi-GPU training.

Takeaways StateX's training speed is comparable to vanilla RNNs, and $2\times$ faster than MoM.

5.5 Parameter Initialization Ablation

Although it may seem natural to inherit pre-trained parameters, our experiments show that reinitializing the modified parameters yields better performance. For Mamba2, StateX introduces new parameters ($\hat{\mathbf{W}}_\mathbf{q}, \hat{\mathbf{W}}_\mathbf{k}$) into the Q and K projections, followed by a joint reinitialization of the expanded projection matrices. In contrast, the inheritance

[6] As works such as Pan et al. (2025) are closed-source and no checkpoints are released, they are excluded from our quantitative comparison.

[7] https://www.github.com/fla-org/flash-linear-attention

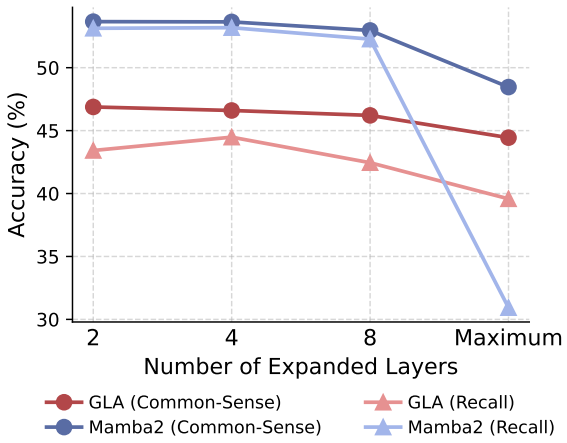

Figure 4: Model performance under varying numbers of expanded layers. Mamba2 has twice as many layers as GLA because it does not have FFN layers. "Maximum" means all layers are expanded.

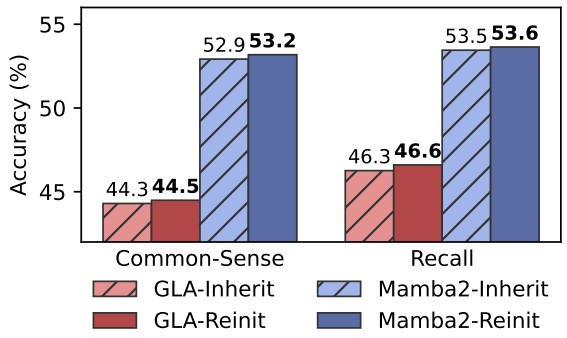

Figure 5: Model performance of reinitialization and parameter inheritance.

strategy for Mamba2 preserves existing parameters while zero-initializing new introduced parameters.

Takeaways As illustrated in Figure 5, the model with reinitialized parameters (Reinit) consistently outperforms the one that inherits parameters (Inherit) on both common-sense reasoning and recall tasks. We hypothesize that the performance gap arises because the inherited parameters have already converged, making it difficult to effectively utilize the newly introduced channels (indicated in red in Figure 1) via post-training.

5.6 Best Proportion of Expanded Layers

As mentioned in Section 4.4, it is important to balance the number of expanded layers. To investigate this trade-off, we conducted an ablation study by varying the number of expanded layers.

Takeaways The results, shown in Figure 4, indicate that both the GLA and Mamba2 models achieve optimal average performance when four layers are expanded (out of 24 layers and 48 layers, respectively). When too many layers are modified, the reinitialized parameters fail to converge effectively under limited continual training, leading to a sharp drop in overall performance.

Head number	CSR ↑	Recall ↑	Train loss ↓
1	**42.7**	**26.0**	**2.722**
4	42.0	24.0	2.762
8	42.4	21.8	2.798
16	41.5	15.4	2.883

Table 4: Common-sense reasoning (CSR), recall, and training loss of GLA-340M models with different numbers of heads. Single-head GLA outperforms other configurations due to larger states.

5.7 The Optimality of Single-Head GLA

As mentioned in Section 4.1, the multi-head mechanism in GLA significantly reduces the size of the recurrent state, which in turn leads to a degradation in model performance. This section presents an ablation study on the number of heads for GLA models trained from scratch. Each GLA model has 340M parameters and is trained on 20B tokens from the SlimPajama dataset (Soboleva et al., 2023) (see Section C.5 for more details). Table 4 reports the performance of these models on a range of common tasks. As shown, the single-head model achieves higher average scores on the benchmark tasks and converges to a lower final training loss.

Takeaway Given the same configurations and number of parameters, fewer heads allow a larger state size, leading to improved performance in common-sense reasoning, recall, and training loss.

6 Conclusions

We have proposed StateX, a novel method for enhancing the recall abilities of two popular RNN variants (linear attention and state space models) by expanding the state sizes of pre-trained RNNs through post-training. Compared to training RNNs with larger state sizes from scratch, our method has much lower training costs and can be seamlessly applied to existing pre-trained models of said RNN variants. StateX is valuable for closing the gap in the recall abilities of RNNs and Transformers. This work represents an important step toward RNNs as an efficient alternative to architectures based on softmax attention.

Limitations

Our work focuses on pure RNN architectures, but, at the moment, hybrid architectures that combine RNN and attention layers have gained traction. The effectiveness of StateX on hybrid architectures is yet to be validated. In addition, we have applied only StateX to the models with 1.3B parameters. Applying StateX to models with tens or hundreds of billions of parameters is interesting, but models with 1.3B parameters are valuable to many practitioners and researchers.

References

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, James Zou, Atri Rudra, and Christopher Ré. 2024a. Simple linear attention language models balance the recall-throughput tradeoff. In *Proceedings of the 41st International Conference on Machine Learning*, pages 1763–1840.

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

Aviv Bick, Eric Xing, and Albert Gu. 2025. Understanding the skill gap in recurrent language models: The role of the gather-and-aggregate mechanism. *Preprint*, arXiv:2504.18574.

Yingfa Chen, Xinrong Zhang, Shengding Hu, Xu Han, Zhiyuan Liu, and Maosong Sun. 2025. Stuffed mamba: Oversized states lead to the inability to forget. *Preprint*, arXiv:2410.07145.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *Preprint*, arXiv:1409.1259.

Tri Dao and Albert Gu. 2024. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning*, pages 10041–10071. PMLR.

Jusen Du, Weigao Sun, Disen Lan, Jiaxi Hu, and Yu Cheng. 2025. Mom: Linear sequence modeling with mixture-of-memories. *Preprint*, arXiv:2502.13685.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, and 5 others. 2024. The language model evaluation harness.

Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. 2025. How to train long-context language models (effectively). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7376–7399, Vienna, Austria. Association for Computational Linguistics.

Albert Gu and Tri Dao. 2024. Mamba: Linear-time sequence modeling with selective state spaces. *Preprint*, arXiv:2312.00752.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9(8):1735–1780.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. 2024. Ruler: What's the real context size of your long-context language models? *Preprint*, arXiv:2404.06654.

Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc V. Le. 2022. Transformer quality in linear time. *Preprint*, arXiv:2202.10447.

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

Samy Jelassi, David Brandfonbrener, Sham M Kakade, and Eran Malach. 2024b. Repeat after me: Transformers are better than state space models at copying. In *International Conference on Machine Learning*, pages 21502–21521. PMLR.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. *Preprint*, arXiv:2006.16236.

Kai Liu, Jianfei Gao, and Kai Chen. 2025. Scaling up the state size of RNN LLMs for long-context scenarios. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11516–11529, Vienna, Austria. Association for Computational Linguistics.

Xingtai Lv, Youbang Sun, Kaiyan Zhang, Shang Qu, Xuekai Zhu, Yuchen Fan, Yi Wu, Ermo Hua, Xinwei Long, Ning Ding, and Bowen Zhou. 2025. Technologies on effectiveness and efficiency: A survey of state spaces models. *Preprint*, arXiv:2503.11224.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. 2022. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11048–11064, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

MiniMax, Aonian Li, Bangwei Gong, Bo Yang, Boji Shan, Chang Liu, Cheng Zhu, Chunhao Zhang, Congchao Guo, Da Chen, Dong Li, Enwei Jiao, Gengxin Li, Guojun Zhang, Haohai Sun, Houze Dong, Jiadai Zhu, Jiaqi Zhuang, Jiayuan Song, and 71 others. 2025. Minimax-01: Scaling foundation models with lightning attention. *Preprint*, arXiv:2501.08313.

Yuqi Pan, Yongqi An, Zheng Li, Yuhong Chou, Ruijie Zhu, Xiaohui Wang, Mingxuan Wang, Jinqiao Wang, and Guoqi Li. 2025. Scaling linear attention with sparse state expansion. *Preprint*, arXiv:2507.16577.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, Kranthi Kiran GV, Xuzheng He, Haowen Hou, Jiaju Lin, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartlomiej Koptyra, Hayden Lau, and 15 others. 2023. Rwkv: Reinventing rnns for the transformer era. *Preprint*, arXiv:2305.13048.

Bo Peng, Daniel Goldstein, Quentin Anthony, Alon Albalak, Eric Alcaide, Stella Biderman, Eugene Cheah, Xingjian Du, Teddy Ferdinan, Haowen Hou, Przemysław Kazienko, Kranthi Kiran GV, Jan Kocoń, Bartłomiej Koptyra, Satyapriya Krishna, Ronald McClelland Jr., Jiaju Lin, Niklas Muennighoff, Fares Obeid, and 11 others. 2024. Eagle and finch: Rwkv with matrix-valued states and dynamic recurrence. *Preprint*, arXiv:2404.05892.

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. 2025. Rwkv-7 "goose" with expressive dynamic state evolution. *Preprint*, arXiv:2503.14456.

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

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. 2021. Linear transformers are secretly fast weight programmers. *Preprint*, arXiv:2102.11174.

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. `https://cerebras.ai/blog/slimpajama-a-627b-token-cleaned-and-deduplicated-version-of-redpajama`.

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

Falcon-LLM Team. 2024. The falcon 3 family of open models.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, Wentao Li, Enzhe Lu, Weizhou Liu, Yanru Chen, Weixin Xu, Longhui Yu, Yejie Wang, Yu Fan, Longguang Zhong, and 41 others. 2025. Kimi linear: An expressive, efficient attention architecture. *Preprint*, arXiv:2510.26692.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need. *Preprint*, arXiv:1706.03762.

Roger Waleffe, Wonmin Byeon, Duncan Riach, Brandon Norick, Vijay Korthikanti, Tri Dao, Albert Gu, Ali Hatamizadeh, Sudhakar Singh, Deepak Narayanan, Garvit Kulshreshtha, Vartika Singh, Jared Casper, Jan Kautz, Mohammad Shoeybi, and Bryan Catanzaro. 2024. An empirical study of mamba-based language models. *Preprint*, arXiv:2406.07887.

Ke Alexander Wang, Jiaxin Shi, and Emily B. Fox. 2025. Test-time regression: a unifying framework for designing sequence models with associative memory. *Preprint*, arXiv:2501.12352.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. 2025. Gated delta networks: Improving mamba2 with delta rule. *Preprint*, arXiv:2412.06464.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. 2024. Gated linear attention transformers with hardware-efficient training. In *International Conference on Machine Learning*, pages 56501–56523.

Tianyuan Zhang, Sai Bi, Yicong Hong, Kai Zhang, Fujun Luan, Songlin Yang, Kalyan Sunkavalli, William T. Freeman, and Hao Tan. 2025. Test-time training done right. *Preprint*, arXiv:2505.23884.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. 2024. ∞bench: Extending long context evaluation beyond 100k tokens. *Preprint*, arXiv:2402.13718.

A Formulation of Gated Linear Attention and Mamba2

For completeness, we provide the complete formulation of GLA and Mamba2 in this section. By default, we use row-vector representation and bold lowercase letters refer to vector values, bold capitalized letters refer to matrix values, while non-bold symbols are scalars.

These models are trained on the next-token prediction task, which means that their input is a sequence of token IDs and their output is a sequence of probability distributions over the vocabulary $\{1, \cdots, V\}$, where V is the vocabulary size.

At the beginning, each token ID is converted to a d-dimensional token embedding by looking up an embedding table (also called the *input embeddings*) before passing to the backbone network. Let T denote the sequence length. This creates a sequence of T embeddings $\mathbf{X}^{(0)} \in \mathbb{R}^{T \times d}$. On the output side, the output embeddings at each position $t \in \{1, \cdots, T\}$ are converted to a probability distribution over the vocabulary via a linear layer called the *language modeling head*.

In the following discussion, we denote the input and output sequences of representations for the l-th layer as:

$$\mathbf{X}^{(l)} = \begin{bmatrix} \mathbf{x}_1^{(l)} \\ \vdots \\ \mathbf{x}_T^{(l)} \end{bmatrix}, \mathbf{Y}^{(l)} = \begin{bmatrix} \mathbf{y}_1^{(l)} \\ \vdots \\ \mathbf{y}_T^{(l)} \end{bmatrix} \tag{6}$$

where T is the sequence length, and $\mathbf{x}_t^{(l)}, \mathbf{y}_t^{(l)} \in \mathbb{R}^{1 \times d}$ are the input and output representations at time step t for the l-th layer. Since the input of each layer is the output of the previous layer, we have $\mathbf{x}_t^{(l)} = \mathbf{y}_t^{(l-1)}$. We use row-representation for all vectors, unless specified.

A.1 Gated Linear Attention

The entire model of GLA consists of interleaving GLA blocks and FFN blocks.

$$\begin{aligned} \mathbf{Y}'^{(l)} &= \mathrm{GLA}^{(l)}\left(\mathbf{X}^{(l)}\right) + \mathbf{X}^{(l)} \\ \mathbf{Y}^{(l)} &= \mathrm{FFN}^{(l)}\left(\mathbf{Y}'^{(l)}\right) + \mathbf{Y}'^{(l)} \end{aligned} \tag{7}$$

Each GLA block consists of multiple heads that are computed in parallel, and the block's output is the sum of the head outputs. This can be formulated as (omitting the layer index for simplicity):

$$\mathbf{y}_t = \sum_{h=1}^{H} \mathrm{GLA}_h(\mathbf{x}_t) \tag{8}$$

Each head in GLA can be formulated as:

$$\begin{aligned} \square_{t,h} &= \mathbf{x}_t \mathbf{W}_{\square,h}, \quad \square \in \{\mathbf{q}, \mathbf{k}, \mathbf{v}, \boldsymbol{\alpha}\}, \\ \mathbf{S}_{t,h} &= \mathrm{diag}(\boldsymbol{\alpha}_{t,h}) \mathbf{S}_{t-1,h} + \mathbf{k}_{t,h}^\top \mathbf{v}_{t,h}, \\ \mathbf{o}_{t,h} &= \mathrm{Norm}(\mathbf{q}_{t,h} \mathbf{S}_{t,h}) \in \mathbb{R}^{d_v}, \\ \mathbf{r}_t &= \mathrm{SILU}(\mathbf{x}_t \mathbf{W}_r + \mathbf{b}_r) \in \mathbb{R}^{d_v}, \\ \mathbf{y}_{t,h} &= (\mathbf{r}_{t,h} \odot \mathbf{o}_{t,h}) \mathbf{W}_{o,h}^\top \in \mathbb{R}^d, \\ \mathbf{y}_t &= \sum_{h=1}^{H} \mathbf{y}_{t,h}. \end{aligned} \tag{9}$$

where $\mathbf{W}_{q,h}, \mathbf{W}_{k,h}, \mathbf{W}_{\alpha,h} \in \mathbb{R}^{d \times d_k}$, $\mathbf{W}_{v,h}, \mathbf{W}_{r,h}, \mathbf{W}_{o,h} \mathbb{R}^{d \times d_v}$ are learnable parameters, SILU is an activation function, and Norm is an RMSNorm layer.

A.2 Mamba2

Mamba2 does not have FFNs and consists only of a stack of Mamba2 blocks:

$$\mathbf{Y}^{(l)} = \mathrm{Mamba2}^{(l)}\left(\mathbf{X}^{(l)}\right) + \mathbf{X}^{(l)} \tag{10}$$

Mamba2 also employs a multi-head mechanism where the layer output is the sum of the head outputs (omitting the layer index for simplicity):

$$\mathrm{Mamba2}(\mathbf{x}_t) = \sum_{h=1}^{H} \mathrm{Mamba2}_h(\mathbf{x}_t) \tag{11}$$

where H is the number of heads, and h is the head index. Each Mamba2 head can be formulated as:

$$\begin{aligned} \mathbf{v}_{t,h} &= \sigma_v(\mathrm{Conv1D}(\mathbf{x}_t \mathbf{W}_{v,h})) \in \mathbb{R}^{d_v}, \\ \mathbf{k}_t &= \sigma_k(\mathrm{Conv1D}(\mathbf{x}_t \mathbf{W}_k)) \in \mathbb{R}^{d_k}, \\ \mathbf{q}_t &= \sigma_q(\mathrm{Conv1D}(\mathbf{x}_t \mathbf{W}_q)) \in \mathbb{R}^{d_k}, \\ \Delta_{t,h} &= \mathrm{SILU}(\mathbf{x}_t \mathbf{W}_{\Delta,h} + \mathbf{b}_{\Delta,h}) \in \mathbb{R}, \\ \alpha_{t,h} &= \exp(-\Delta_t \exp(A_h)) \in \mathbb{R}, \\ \mathbf{S}_{t,h} &= \mathbf{S}_{t-1,h} \alpha_{t,h} + \Delta_{t,h} \mathbf{k}_t^\top \mathbf{v}_{t,h} \in \mathbb{R}^{d_k \times d_v}, \\ \mathbf{o}_{t,h} &= \mathbf{q}_t \mathbf{S}_{t,h} + D_h \mathbf{v}_{t,h} \in \mathbb{R}^{d_v}, \\ \mathbf{z}_{t,h} &= \mathrm{SILU}(\mathbf{x}_t \mathbf{W}_{z,h}) \in \mathbb{R}^{d_v}, \\ \mathbf{y}_{t,h} &= \mathrm{Norm}(\mathbf{o}_{t,h} \odot \mathbf{z}_{t,h}) \mathbf{W}_{o,h}^\top \in \mathbb{R}^d, \\ \mathbf{y}_t &= \sum_{h=1}^{H} \mathbf{y}_{t,h}. \end{aligned} \tag{12}$$

where Conv1D is a per-channel 1-dimensional convolutional layer with a kernel size of 4, $\mathbf{W}_{\Delta,h} \in \mathbb{R}^{d \times 1}, \mathbf{b}_{\Delta,h}, A_h \in \mathbb{R}, \mathbf{W}_{v,h}, \mathbf{W}_{z,h}, \mathbf{W}_{o,h} \in \mathbb{R}^{d \times d_v}, \mathbf{W}_k, \mathbf{W}_q \in \mathbb{R}^{d \times d_k}$ are learnable parameters.

Model	Update rule	Query rule	State size	StateX state size
GLA	$\mathbf{S}_{t-1,h}\text{diag}(\alpha_{t,h}) + \mathbf{k}_{t,h}^T\mathbf{v}_{t,h}$	$\mathbf{q}_{t,h}\mathbf{S}_{t,h}$	Hd_kd_v	$H^2d_kd_v$
Mamba2	$\mathbf{S}_{t-1,h}\alpha_{t,h} + \Delta_{t,h}\mathbf{k}_t^T\mathbf{v}_{t,h}$	$\mathbf{q}_t\mathbf{S}_{t,h} + D_h\mathbf{v}_{t,h}$	Hd_kd_v	Hd_vd_kE

Table 5: Overview of GLA and Mamba2, two popular RNNs with matrix-valued recurrent states. H, P, N, d_k, d_v are hyperparameters of the architectures. E is the expansion ratio of StateX for SSMs, which is set to 4, as mentioned in Section 4.2

Model	Params	Total State	SWDE	SQuAD	TQA	NQ	Drop	Avg. ↑
Linear Attention — GLA								
Vanilla	1.365B	12.58M	47.1	56.8	56.0	21.9	36.5	43.7
StateX (ours)	1.365B	18.87M	50.3	59.1	55.0	21.8	39.5	**45.1**
State Space Model — Mamba2								
Vanilla	1.343B	24.96M	54.1	57.8	63.5	36.8	35.4	49.5
StateX (ours)	1.350B	37.44M	56.1	57.9	63.6	36.4	36.3	**50.1**
Sparse Model – MoM								
MoM (Du et al., 2025)	1.552B	31.45M	34.4	49.6	50.1	16.0	33.9	36.8

Table 6: Accuracy on recall-intensive tasks with sequences truncated to a maximum of 2K tokens, as well as the model size and state size of each model. The best scores are bolded.

Models	8-shot ↑	16-shot ↑	24-shot ↑
Linear Attention—GLA			
Vanilla	47.3	49.7	48.4
StateX (ours)	**48.1**	**52.4**	**51.9**
State Space Models—Mamba2			
Vanilla	**47.7**	49.7	52.4
StateX (ours)	47.6	**52.3**	**53.0**
Sparse Model – MoM (Du et al., 2025)			
MoM	42.6	42.2	42.9

Table 7: In-context learning performance of GLA and Mamba2 variants, evaluated on 12 downstream classification tasks. Higher is better.

A.3 Update Rule and Query Rule

Central to recurrent architectures are the update rule and query rule (described in Section 3.1), which dictate how the architecture models inter-token dependencies. Table 5 shows the update rule and query rule of GLA and Mamba2.

A.4 Details of Parameter Reinitialization

In the case of GLA, we reinitialize all parameters within the GLA block, including its normalization layer. For Mamba2, StateX reinitializes all parameters of $A_h, \mathbf{W}'_k, \mathbf{W}'_q$. Moreover, we reinitialize the bias term of the projection layer for the discretization term ($\Delta_{t,h}$), which is called `dt_bias` in the official implementation of Mamba2[8].

B More Results

B.1 Detailed Results on Recall, ICL, and Common-Sense Reasoning

Tables 6, 7, and 8 reports the details results of the models studied in this paper in each subtask measuring recall abilities, in-context learning, and common-sense reasoning.

B.2 Improvement on Long-Context Retrieval

The recall-intensive tasks we used in Section 5.2 contain mostly sequences with fewer than 4K tokens. To evaluate the models' abilities to retrieve information from longer contexts, we use the popular NIAH task (Hsieh et al., 2024). Due to differences in the recall abilities between the GLA and Mamba2, we evaluate them using NIAH tasks of varying difficulty to avoid score saturation and preserve discriminative resolution. For the GLA model, we employed the simpler passkey retrieval task from ∞Bench (Zhang et al., 2024), which involves retrieving a single 5-digit passkey from long documents consisting of repeated text. For Mamba2, we use the more challenging NIAH-Single-2 task from RULER (Hsieh et al., 2024), where a 7-digit passkey is embedded in a semantically meaningful, non-repetitive distractor content.

[8] https://github.com/state-spaces/mamba/blob/main/mamba_ssm/modules/mamba2.py

Model	PIQA acc ↑	Hella. acc ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc ↑	SIQA acc ↑	Avg. ↑
Linear Attention—GLA							
Vanilla	69.6	38.2	54.7	54.5	22.7	39.6	46.5
StateX (ours)	69.7	37.1	54.9	53.9	22.5	39.9	46.3
State Space Model—Mamba2							
Vanilla	73.0	45.4	59.6	64.3	29.1	41.1	52.1
StateX (ours)	73.6	45.0	59.9	64.0	29.6	41.6	52.3
Sparse Model—MoM							
MoM (Du et al., 2025)	63.3	30.4	50.8	45.2	18.8	37.4	41.0

Table 8: Performance on language modeling and zero-shot common-sense reasoning.

Context Length	4K	8K	16K	32K
GLA—Passkey Retrieval				
Vanilla	0.74	0.41	0.13	0.01
StateX (ours)	**0.93**	**0.77**	**0.34**	**0.06**
Mamba2—NIAH-Single-2				
Vanilla	0.83	0.43	0.30	**0.09**
StateX (ours)	**0.94**	**0.61**	**0.32**	**0.09**

Table 9: Performance on retrieving specific information (i.e., a needle) from synthetically generated long documents up to 64K tokens.

More details can be found in Appendix C.4.

Results Table 9 reports the models' performances in NIAH. It shows that, by unlocking a larger state size, StateX significantly improves the model's recall performance in long contexts.

B.3 More Efficiency Results

Tables 10 and 11 show the detailed inference throughput results of various models studied in this paper.

Ctx. len.	1K	4K	16K	64K	Avg. ↑
Linear Attention — GLA					
GLA	72.5K	72.6K	72.2K	70.4K	71.9K
—StateX	70.0K	70.1K	69.5K	68.0K	69.4K
State Space Model — Mamba					
Mamba2	44.0K	44.1K	44.2K	50.7K	45.7K
—StateX	45.1K	45.3K	42.6K	52.2K	46.3K
RNN with Sparse States — MoM					
MoM	20.3K	28.1K	29.5K	31.4K	27.3K

Table 10: Prefilling throughput (tokens/s) of various models, measured with different context lengths.

BSZ	64	128	256	512	Avg. ↑
Linear Attention—GLA					
GLA	3548	3815	9595	10226	6796
–StateX	3770	6371	7083	7395	6155
State Space Model—Mamba					
Mamba	2276	3034	4755	5730	3949
–StateX	2174	3121	4351	4837	3620
RNN with Sparse State—MoM					
MoM	46.3	47.5	48.2	OOM	47.3

Table 11: Decoding throughput (tokens/s) of various models, measured with different batch sizes.

B.4 Training Loss

We also tracked the training loss curves of models trained with standard continual training and with StateX. Figure 6 shows the loss curves for both GLA and Mamba2. The former has generally lower loss because it was pre-trained on SlimPajama, while Mamba2 was not. Notably, the StateX models have a higher initial training loss due to the architectural change, but quickly close the gap. Interestingly, although their final training loss is slightly higher than the continual training counterparts, they achieve better performance on downstream tasks.

C Experiment Details

C.1 Baselines

The vanilla models are obtained through continual training on the data mentioned before. In contrast, the StateX models incorporate an additional state expansion step before continual training. MoM (Du et al., 2025), a representative sparse state model, is trained from scratch and has undergone the same amount of data throughout continual training, as there are no available pre-trained checkpoints for

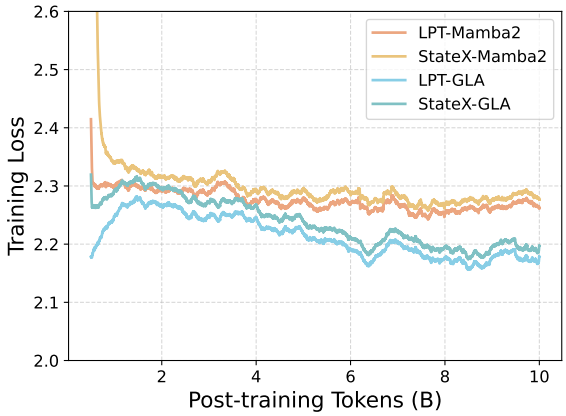

Figure 6: Post-training loss (on SlimPajama) of vanilla models and expanded models. GLA has lower loss as it is pre-trained on SlimPajama while Mamba2 is pre-trained on Pile. LPT means the vanilla models undergoing continual training.

these novel architectures with large states.

C.2 Evaluation

We configure the evaluation tasks using the lm-evaluation-harness framework (Gao et al., 2024). A set of widely adopted benchmark tasks is selected to assess the models' capabilities in common-sense reasoning and information recall. For the common-sense and recall tasks, we adopt *accuracy* (not *normalized accuracy*) and *contains* as the respective evaluation metrics. *Accuracy* directly reflects the correctness of the common-sense task results, while *contains* measures the proportion of recall task outputs that include the passkey. Notably, for tasks related to recall ability, we adopt the "Just Read Twice" prompt from Arora et al. (2024b), which is also used in Yang et al. (2024) and Yang et al. (2025), given that all models under evaluation are based on recurrent architectures.

C.3 In-Context Learning Evaluation

For the in-context learning (ICL) evaluation, we follow the setup introduced by Min et al. (2022), which systematically benchmarks ICL capabilities across classification and multiple-choice tasks. Our evaluation adopts the same protocol, but we also evaluate with a different number of in-context demonstrations for comprehensiveness.

The tasks that were used for evaluation are:

- `commonsense_qa`
- `ai2_arc`
- `superglue-copa`
- `superglue-cb`
- `glue-mrpc`
- `glue-sst2`
- `glue-qqp`
- `glue-cola`
- `superglue-rte`
- `superglue-wic`
- `codah`
- `dream`

C.4 Needle-in-a-Haystack Tasks

As mentioned in the previous section, we design two passkey retrieval tasks with varying levels of difficulty. The specific noise configurations and prompt templates used in each task are detailed in Table 12. We use 5-digit passkeys in Passkey Retrieval and 7-digit passkeys in NIAH-Single-2. For each unique test length, the task will be tested on 256 randomly generated examples to ensure the consistency of the results.

C.5 More Details: Ablation Study on the Number of GLA Heads

The training procedure for these models follows common language model pre-training practices as closely as possible. The model is trained on 20B tokens from SlimPajama, with a 0.5M tokens per batch, and a sequence length of 4k. We employ a cosine learning rate scheduler with an initial learning rate of 3e-4 and no specified minimum learning rate. All models consist of 340 million parameters and comprise 24 layers, each with an identical hidden state dimension. The only architectural difference lies in the number of attention heads: the single-head model uses one head with a dimensionality of 512, while the four-head model uses four heads, each with a dimensionality of 128, and so on, following the same principle.

D The Use of Large Language Models

Large language models (LLMs) were used to quality-check the final draft, but we never explicitly instruct LLMs to write any parts of this paper.

Passkey Retrieval (∞Bench)	**Task Template:** The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again. ..... The pass key is {number}. Remember it. {number} is the pass key. ..... The grass is green. The sky is blue. The sun is yellow. Here we go. There and back again. **Task Answer Prefix:** What is the pass key? The pass key is
NIAH-Single-2 (RULER)	**Task Template:** Some special magic numbers are hidden within the following text. Make sure to memorize it. I will quiz you about the numbers afterwards. Paul Graham Essays. ..... One of the special magic numbers for {word} is: {number}. ..... What is the special magic number for {word} mentioned in the provided text? **Task Answer Prefix:** The special magic number for {word} mentioned in the provided text is

Table 12: The prompt templates of the NIAH tasks used to evaluate the models in retrieving information from long contexts.