FLASHBACK: Memory Mechanism for Enhancing Memory Efficiency and Speed in Deep Sequential Models

Taiki Sekii
CyberAgent
taiki.sekii@gmail.com

Abstract

In this study, we tackle three main challenges of deep sequential processing models in previous research: (1) memory degradation, (2) inaccurate gradient backpropagation, and (3) compatibility with next-token prediction. Specifically, to address (1–2), we define a *Flashback property* in which memory is preserved perfectly as an identity mapping of its stored value in a memory region until it is overwritten by a hidden state at a different time step. We propose a *Flashback mechanism* that satisfies this property in a fully differentiable, end-to-end manner. Further, to tackle (3), we propose architectures that incorporate the Flashback mechanism into Transformers and Mamba, enabling next-token prediction for language modeling tasks. In experiments, we trained on The Pile dataset, which includes diverse texts, to evaluate tradeoffs between commonsense reasoning accuracy, processing speed, and memory usage after introducing the Flashback mechanism into existing methods. The evaluations confirmed the effectiveness of the Flashback mechanism.

1 Introduction

The term "Generative AI" has rapidly gained widespread attention, particularly due to applications such as ChatGPT (OpenAI) and Stable Diffusion (Stability AI). Consequently, various applications, such as chat-based interfacing (OpenAI), image generation (Stability AI), and programming support (GitHub, Inc.), have emerged. Several of these systems are built on foundation models (Bommasani et al., 2021), which can process not only text (Touvron et al., 2023; OpenAI et al., 2024) but also multimodal data such as video and audio (Girdhar et al., 2023; Madan et al., 2024).

For tasks that solve users' problems through natural language processing, as seen with ChatGPT, these systems typically rely on deep neural networks (DNNs), particularly Transformer-based models (Vaswani et al., 2017; Touvron et al., 2023; OpenAI et al., 2024). Such models are highly scalable, especially in terms of handling larger datasets, enabling them to tackle complex tasks that were previously infeasible before 2022.

However, as generative AI advances, a significant limitation arises in terms of handling information not included in the training dataset, such as newly acquired user input or continuously updated databases (referred to as *external memory* (Graves et al., 2014, 2016)). Transformer-based models struggle to retain this external memory beyond a certain capacity due to their architectural characteristics stemming from the computational and memory costs that increase with the length of input sequences in attention mechanisms (Vaswani et al., 2017; Dai et al., 2019). Two general approaches have been explored to address this issue: (1) implementing memory mechanisms on the application side or (2) integrating memory capabilities within the DNN architecture.

The first approach involves enhancing applications with memory capabilities, such as enabling large language model (LLM)-based agents to search databases (Press et al., 2023) or retrieve relevant information using techniques such as retrieval-augmented generation (Lewis et al., 2020) (RAG). The second approach directly incorporates memory mechanisms within DNN architectures, leading to memory-capable neural networks (MC-NNs) (Hochreiter and Schmidhuber, 1997; Graves et al., 2014; Weston et al., 2015; Sukhbaatar et al., 2015; Graves et al., 2016; Rae et al., 2020; Gu et al., 2021; Bulatov et al., 2022; Fu et al., 2023; Gu et al., 2022; Yang et al., 2023; Peng et al., 2023; Gu and Dao, 2023; Arora et al., 2024). These approaches often build upon recurrent neural networks (RNNs), which store input information in hidden states or extend them with explicit memory components known as memory-augmented neural networks (MANNs).

RNN-based methods offer the advantage of not

Proceedings of the 31st International Conference on Computational Linguistics, pages 8602–8611
January 19–24, 2025.

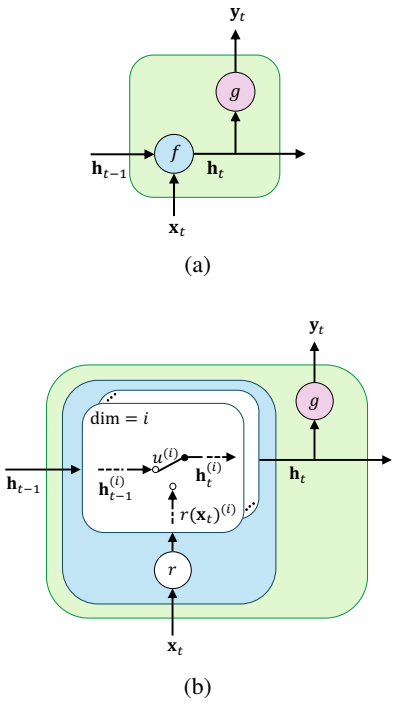

Figure 1: Conceptual illustration of (a) a basic RNN and (b) the Flashback mechanism.

requiring a custom memory architecture for each application. Recently, subquadratic models that do not impose limitations on sequence length have garnered attention, particularly state-space models (Gu et al., 2021, 2022; Fu et al., 2023; Gu and Dao, 2023) (SSMs) and linear attention-based models (Yang et al., 2023; Peng et al., 2023; Arora et al., 2024). Further, efforts to improve the efficiency of attention computation in Transformers for long input sequences have been made (Parmar et al., 2018; Child et al., 2019; Beltagy et al., 2020; Sun et al., 2023; Ding et al., 2023). Owing to their scalability in handling large datasets through self-supervised learning, both Transformer- and RNN-based methods are becoming the backbones for foundation models used across diverse applications. However, achieving higher precision, faster processing, and less memory usage remains challenging, especially for more advanced applications, such as running LLMs on mobile devices or robots and processing multimodal data that includes text, video, and audio. In this study, we focus on enhancing the efficiency of RNN-based methods that can handle inputs with no constraints on sequence length.

In previous studies on MCNNs, the following challenges have been identified when improving performance and scalability for practical applications.

1. **Memory Degradation.** Memory regions, such as hidden states or explicit memory components, must be updated regularly at specific intervals to retain external memory. However, it is difficult to completely prevent memory degradation over time, whether due to incorrect updates or continuous alterations.

2. **Inaccurate Gradient Backpropagation.** Similar to the issues identified with RNNs, architectures that continuously update memory suffer from vanishing or exploding gradients when backpropagating through long sequences because the gradients must be propagated over many time steps.

3. **Incompatibility with Next-Token Prediction.** When memory is updated at regular intervals or based on segments of input rather than at every time step, the architecture cannot be seamlessly applied to next-token prediction, a common pretraining task for modern LLMs.

1.1 Overview

In this study, we tackle all three challenges by introducing a novel memory mechanism, the FLASHBACK *mechanism*, which retains memory *sharply* and can be incorporated into next-token prediction architectures without modification. The FLASHBACK *property* ensures that once external memory is stored, it is retained as an identity mapping until explicitly overwritten. This allows gradients to be backpropagated directly from later time steps in a fully differentiable, end-to-end manner when the memory is referenced.

In the proposed Flashback mechanism, memory regions are represented as hidden states, similar to RNNs. At each time step, the current hidden state is compared with the input data and elements in the memory region are replaced if necessary to satisfy the Flashback property, as shown in Table 1. Finally, we propose architectures that integrate the Flashback mechanism into both Transformers and Mamba, thereby allowing them to handle next-token prediction for language modeling tasks.

The effectiveness of the Flashback property was evaluated through experiments on The Pile (Gao et al., 2020), focusing on language modeling tasks. We assessed the changes in processing accuracy, speed, and memory usage after incorporating the

Flashback mechanism into existing methods to demonstrate its effectiveness.

The major contributions of this study are twofold: (1) defining a Flashback property to approach the three challenges of existing MCNNs and (2) proposing a Flashback mechanism that can be integrated into next-token prediction architectures such as LLMs.

2 Related Work

2.1 Early MCNNs

MCNNs for handling sequential data have been extensively studied, and numerous approaches have been proposed, including RNNs and long-short-term memory (LSTM) networks (Rumelhart et al., 1986; Hochreiter and Schmidhuber, 1997). Early RNN-based methods store external memory solely within the hidden states of neural networks at each time step. However, these models struggle to account for dependencies between temporally distant input data. In response to this limitation, methods based on MANNs, which extend RNN architectures, have been proposed in recent years (Graves et al., 2014; Weston et al., 2015; Sukhbaatar et al., 2015; Graves et al., 2016). Notably, neural Turing machines (Graves et al., 2014) and differentiable neural computers (DNCs) (Graves et al., 2016) introduce explicit memory regions distinct from hidden states in neural network architectures, allowing for direct read and write operations to memory stores.

2.2 Transformer-Based Approaches

As discussed in Section 1, the high generalizability gained from large-scale training datasets has led to the widespread adoption of Transformers, particularly for language modalities. Consequently, several methods that extend the MANN concept by explicitly integrating external memory into the Transformer architecture have been proposed (Rae et al., 2020; Bulatov et al., 2022). Such methods enable the handling of long-range input sequences by referencing past information from memory regions while maintaining inference efficiency that remains constant with respect to the input sequence length. In addition, techniques have been developed in application-layer solutions outside DNNs, where pretrained LLMs function as agents that query databases (Press et al., 2023) or use methods such as RAG (Lewis et al., 2020) to extract relevant information from external sources.

2.3 Backbone Architectures

Enhancing the performance of core DNNs that serve as the backbone for foundation models could allow them to handle longer input sequences, eliminating the need for introducing additional memory regions into either the DNN architecture or application layer. In line with this approach, several methods that improve attention mechanisms to efficiently process long-range input data have been proposed (Parmar et al., 2018; Child et al., 2019; Beltagy et al., 2020; Sun et al., 2023; Ding et al., 2023). Such methods broaden and sparsify the relationships modeled by attention mechanisms, enabling attention computation between temporally distant inputs. However, this approach inherently limits the maximum sequence length that can be processed by DNNs.

Approaches based on SSMs (Gu et al., 2021, 2022; Fu et al., 2023; Gu and Dao, 2023) and linear attention models (Yang et al., 2023; Peng et al., 2023; Arora et al., 2024), which are inspired by RNN architectures, store external memory in hidden states without imposing limitations on the input sequence length. Such RNN-based methods not only exhibit generalizability comparable to Transformers, especially in problem settings with relatively few weight parameters, but also have the advantage of inference efficiency independent of sequence length.

This study draws inspiration from certain MANN approaches that directly store input vectors in memory regions (Weston et al., 2015; Sukhbaatar et al., 2015). Unlike these methods, we selectively stores each element of the input vector as an identity mapping, enabling a fully differentiable, end-to-end process.

3 Proposed Method

In this section, we first explain the principles of RNNs and clarify the challenges faced in previous studies. Next, we define the Flashback property, propose the Flashback mechanism that satisfies this property, and present a DNN architecture that incorporates the Flashback mechanism.

3.1 Challenges in RNN-Based Approaches

In RNNs, the hidden state at time t, $\mathbf{h}_t \in \mathbb{R}^D$, is computed from the hidden state at time $t-1$, $\mathbf{h}_{t-1}$, and the input at time t, $\mathbf{x}_t \in \mathbb{R}^D$, as follows:

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t), \tag{1}$$

where $f(\cdot)$ is the state update function. The output of the RNNs at time t, $\mathbf{y}_t$, is computed using $\mathbf{h}_t$:

$$\mathbf{y}_t = g(\mathbf{h}_t), \tag{2}$$

where $g(\cdot)$ denotes the output function. Thus, the hidden state at time $t+T$, $\mathbf{h}_{t+T}$, can be expressed recursively as follows:

$$\begin{aligned}\mathbf{h}_{t+T} =& f(f(f(\cdots f(\mathbf{h}_{t-1}, \mathbf{x}_t) \\ & \cdots), \mathbf{x}_{t+T-1}), \mathbf{x}_{t+T}).\end{aligned} \tag{3}$$

Because $\mathbf{h}_t$ is recursively transformed by $f(\cdot)$ until time $t+T$, there is a risk of memory degradation due to incorrect transformations or error accumulation (see Challenge 1 in Section 1). At this point, the gradient of loss at time $t+T$, L_{t+T}, with respect to hidden state $\mathbf{h}_t$ can be computed using backpropagation as follows:

$$\frac{\partial L_{t+T}}{\partial \mathbf{h}_t} = \frac{\partial L_{t+T}}{\partial \mathbf{h}_{t+T}} \cdot \frac{\partial \mathbf{h}_{t+T}}{\partial \mathbf{h}_{t+T-1}} \cdot \cdots \cdot \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_t}. \tag{4}$$

Note that in next-token prediction tasks, the loss L is computed at every time step. The multiplication of several derivatives in this chain rule introduces the risk of vanishing or exploding gradients (see Challenge 2 in Section 1).

When external memory is stored in a hidden state, many prior works on MANNs, although complex, fundamentally adhere to the basic principles of RNNs. For example, operations on memory matrices using a controller in DNCs (Graves et al., 2016) or write operations to memory tokens in RMTs (Bulatov et al., 2022) can be interpreted as updates to the hidden state via $f(\cdot)$. In addition, methods based on SSMs or linear attention are designed based on RNN concepts.

3.2 Flashback Property

In the proposed method, the computation of hidden state $\mathbf{h}_t$ can be performed as follows:

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}) = \mathrm{Update}(r(\mathbf{x}_t), \mathbf{h}_{t-1}), \tag{5}$$

where $r(\mathbf{x}_t) \in \mathbb{R}^D$ denotes a function that transforms $\mathbf{x}_t$ either linearly or nonlinearly. The function $\mathrm{Update}(r(\mathbf{x}_t), \mathbf{h}_{t-1})$ updates the i-th element of $\mathbf{h}_t$ by selecting either the corresponding element of $r(\mathbf{x}_t)$ or $\mathbf{h}_{t-1}$ as follows:

$$\mathbf{h}_t^{(i)} = \begin{cases} r^{(i)}(\mathbf{x}_t) & \text{if } u^{(i)} = 1 \\ \mathbf{h}_{t-1}^{(i)} & \text{if } u^{(i)} = 0 \end{cases}, \tag{6}$$

where $u^{(i)}$ denotes a flag determined by a predefined criterion for each element of $\mathbf{h}_t$ based on the $\mathrm{Update}(\cdot)$ function.

In this study, we focus on a property where each element of $\mathbf{h}_t$ becomes an identity mapping of the elements of previous inputs or hidden states. When this property is satisfied, the hidden states are maintained at the updated value, reducing the risk of memory degradation due to recursive transformations described in Section 1. Further, if the i-th element of the hidden state at time $t+T$, $\mathbf{h}_{t+T}^{(i)}$, retains the value from time t,

$$\frac{\partial \mathbf{h}_{t+T}^{(i)}}{\partial \mathbf{h}_t^{(i)}} = 1 \tag{7}$$

holds, meaning that the gradient of the loss at time $t+T$, $\partial L_{t+T}/\partial \mathbf{h}_t^{(i)}$, can be computed without the multiplication of derivatives described in Equation (4). As a result, the risk of vanishing or exploding gradients in RNNs is significantly reduced. We refer to this property, which mitigates the issues of RNNs, as the Flashback property. By maintaining the direct forward and backward propagation of memory information and its gradients over long periods, we aim to promote efficient memorization and stable learning.

3.3 Flashback Mechanism

In this study, we propose the Flashback mechanism, a memory mechanism that possesses the Flashback property. Its architecture is shown in Figure 1. First, hidden state $\mathbf{h}_t$ is updated using the linear transformation $r(\mathbf{x}_t)$ of $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$ as follows:

$$\mathbf{h}_t = \mathrm{MaxPool}(r(\mathbf{x}_t); \mathbf{h}_{t-1}), \tag{8}$$

where Max-Pooling is adopted as the simplest $\mathrm{Update}(\cdot)$ function that satisfies the Flashback property, and $\mathrm{MaxPool}(\cdot)$ selects the maximum value across the elements in the channel dimension. Next, the output $\mathbf{y}_t$ is obtained as follows[1]:

$$\begin{aligned}\mathbf{z}_t =& \mathrm{LayerNorm}(\mathbf{x}_t \\ & + \mathrm{GeLU}(q(\mathbf{x}_t) + \mathrm{Norm}(\mathbf{h}_{t-1}))),\end{aligned} \tag{9}$$

$$\mathbf{y}_t = \mathrm{LayerNorm}(\mathbf{z}_t + \mathrm{FFN}(\mathbf{z}_t)), \tag{10}$$

where $\mathrm{LayerNorm}(\cdot)$ represents layer normalization, $\mathrm{GeLU}(\cdot)$ is the Gaussian error linear unit

[1]Although the definition from input to output function does not exactly match the simplified RNN formulation in Section 3.2, it is equivalent in principle.

function, $q(\cdot)$ is a linear transformation function, $\mathrm{Norm}(\cdot)$ is a normalization function, and $\mathrm{FFN}(\cdot)$ denotes feed forward networks (FFNs).

3.4 Flashback DNN Architecture

In this study, we propose DNN architectures that integrate the Flashback mechanism into both Transformers (Vaswani et al., 2017) and Mamba (Gu and Dao, 2023). For the Transformer architecture, we use Transformer++ (Touvron et al., 2023) (LLaMa). The attention mechanism and FFNs are treated as a single block. For Mamba, the SSM architecture is considered a single block. Figure 2 shows the architectures of Transformer++ and Mamba integrated with the Flashback mechanism. The Flashback mechanism is inserted at every even-numbered block in the architectures to introduce the Flashback property.

In the Transformer-based architecture, we employ sliding window attention (Parmar et al., 2018; Child et al., 2019; Beltagy et al., 2020) (SWA) to keep computational complexity within the sub-quadratic range, ensuring that processing efficiency does not degrade as input sequence length increases. The embedding dimensions for input tokens and hidden states in the Flashback mechanism are set according to the specific configurations of Transformer++ and Mamba.

4 Experiments

In this section, we describe the experimental configuration in detail and then verify the effectiveness of the proposed method from the following three perspectives.

- Evaluate the language modeling performance of the proposed method in the context of commonsense reasoning tasks.
- Assess the generalizability of the proposed method when applied to input data that exceed the learned sequence length.
- Analyze the effectiveness and characteristics of the Flashback property through multiple ablation studies.

4.1 Datasets

For the experiments, we used The Pile dataset (Gao et al., 2020) for training and the LM-Eval harness tool (Gao et al., 2024) for evaluating language modeling accuracy.

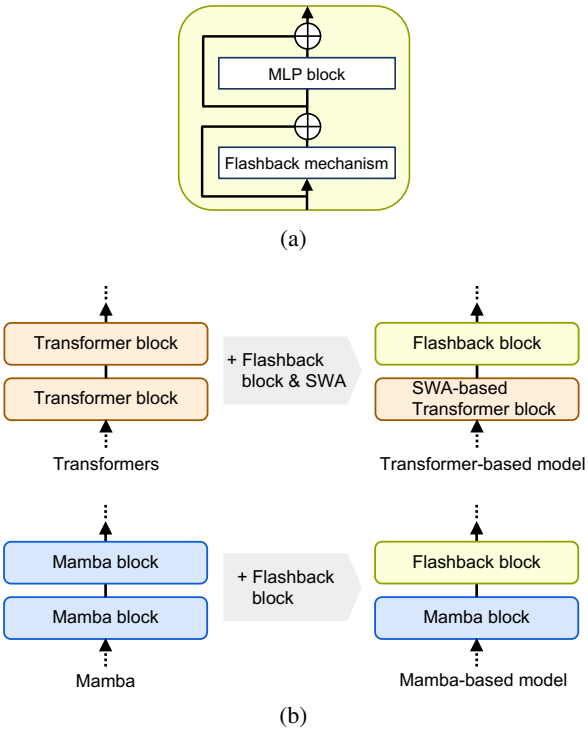

Figure 2: An overview of the proposed architecture design. (a) The Flashback block is integrated into (b) the entire architecture.

4.1.1 Training

The Pile is a large collection of high-quality text gathered from various sources and has been widely used in recent research on deep sequential models. Specifically, it comprises 22 high-quality subsets, which total approximately 800 GB. The subsets include text from academic papers, books, programs, and technical documentation, enabling the evaluation of language models' generalizability across diverse text domains.

4.1.2 Testing

The LM-Eval harness, provided by EleutherAI, is a standardized benchmark that covers various datasets, including the following.

- LAMBADA (Paperno et al., 2016): A dataset that requires models to understand the entire context and predict the final word. This dataset evaluates the ability to consider multiple sentences rather than just individual ones.
- HellaSwag (Zellers et al., 2019): A dataset that measures commonsense reasoning by asking models to choose the most appropriate option to complete a description of everyday scenarios.

- PIQA (Bisk et al., 2020): A dataset that assesses physical commonsense reasoning by requiring models to choose the correct solution for everyday situations.

- ARC (Clark et al., 2018): A dataset focused on scientific commonsense and reasoning, divided into *Challenge* and *Easy* subsets, each containing scientific problems of varying difficulty.

- WinoGrande (Sakaguchi et al., 2019): A dataset for resolving pronoun references based on context, thereby testing the model's ability to understand contextual relationships.

4.2 Experimental Setup

We implemented the baseline methods and the proposed method based on the publicly available implementation of BASED (Arora et al., 2024). Both the baseline and proposed methods were trained on the same sequence of 10B tokens from The Pile. The text was tokenized using the GPT-2 BPE tokenizer (Radford et al., 2019). For attention-based methods, FlashAttention-2 (Dao et al., 2022; Dao, 2024) was used for both training and testing. The batch sizes during training were adjusted for each method to maximize memory usage, and the sequence length for training was set to 2K tokens across all methods.

In the proposed method, we evaluated two model sizes for Transformer++ and Mamba: approximately 360M and 1.3B parameters. The window size for the sliding window attention in the Transformer-based Flashback architecture was set to 512, which is one-fourth of the sequence length used during training. The total number of blocks in the proposed architecture was adjusted so that the token prediction accuracy (perplexity, ppl.) remained consistent before and after integrating the Flashback mechanism (see Section 4.5.3). Tables 1 and 2 list the hyperparameters employed by the proposed methods. To ensure a fair performance comparison, these configurations remain unchanged from the conventional methods (Arora et al., 2024), except for the number of blocks.

4.2.1 Evaluation Metrics

For evaluation metrics, we used the same criteria as in (Arora et al., 2024); the details can be found in their literature. To ensure fair comparison across methods, all training and testing were conducted using a common NVIDIA A100 GPU. Token generation throughput (tokens per millisecond, Tok./ms) was used to compare the processing speeds of each method. During speed measurements, batch sizes were adjusted to maximize throughput while remaining within feasible execution limits for each method. This adjustment ensures fair comparison by allowing each method to utilize its full potential, accounting for differences in computational characteristics and hardware efficiency. Memory efficiency was compared using the total GPU memory usage during speed measurements, divided by the batch size (MiB/Seq.).

Following previous studies (Maddox et al., 2020; Curth et al., 2023), rather than relying solely on parameter counts as a measure of model complexity, we evaluate each method's vanilla architecture on standard hardware in terms of raw computational performance, without employing model quantization or introducing additional inductive biases.

Base architecture param.	360M	1.3B
Optimizer	Adam	
Optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.95$	
Optimizer eps	$1e-8$	
Precision	BFloat16	
Warmup	1%	
Learning rate decay	Cosine	
Learning rate (min, base)	$8e-5, 8e-4$	
Global batch size	256	
Weight decay	0.1	
# blocks	36	64
Hidden size	1024	1680
# heads	16	24
RMSNorm	True	
MLP bias	False	
Rotary emb. fraction	0.5	
MLP activation	SwiGLU	
MLP width	4	

Table 1: Training configurations and hyperparameters for the Transformer-based model.

Base architecture param.	360M	1.3B
Optimizer	Adam	
Optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.95$	
Optimizer eps	$1e-8$	
Precision	BFloat16	
Warmup	1%	
Learning rate decay	Cosine	
Learning rate (min, base)	$8e-5, 8e-4$	
Global batch size	256	
Weight decay	0.1	
# blocks	42	
Hidden size	1024	2048
RMSNorm	True	
Norm epsilon	$1e-5$	
Dt State	16	
Dt (min, max)	$(0.001, 0.1)$	
Dt init. strategy	Random	
Dt init. floor	$1e-4$	
Dt scale	1.0	
Dt softplus	True	
Projection expansion factor	2	
Short conv. filter size	4	

Table 2: Training configurations and hyperparameters for the Mamba-based model.

We compare the proposed method with several baseline methods from previous research in terms of processing efficiency. Transformer++ is a Transformer-based model with rotary encoding for positional embeddings and gated linear units. Mamba is a state-of-the-art method using SSMs that selectively store important information in hidden states depending on the input. In addition, we include early DNN architectures based on convolutional approaches, such as H3 (Fu et al., 2023), RWKV (Peng et al., 2023), and GLA (Yang et al., 2023), as comparison targets.

4.3 Evaluation on Commonsense Reasoning

The results of comparing the accuracy of each method in commonsense reasoning language modeling tasks are shown in Table 3. The results of comparing the accuracy, processing speed, and memory usage across methods are shown in Figure 3. From Table 3, integrating the Flashback mechanism into Transformer++ and Mamba reduced the number of attention and SSM blocks, respectively, while maintaining reasoning accuracy (see Section 4.5.3). Meanwhile, Figure 3 shows that integrating the Flashback mechanism into 1.3B-parameter Transformer++ and using sliding window attention resulted in more than a fourfold speed improvement and a 77% memory usage reduction. For RNN-based methods, incorporating the Flashback mechanism into 1.3B-parameter Mamba led to a 16% speed improvement and a 38% memory usage reduction. Based on these results, the Flashback mechanism is effective as a memory mechanism for next-token prediction, thereby addressing Challenges 1–3.

4.4 Evaluation on Long-Sequence Input Data

We compared the next-token prediction accuracy, processing speed, and memory usage for long-sequence input data from The Pile across the baseline and proposed methods. The results are shown in Table 4. From the table, we observe that accuracy remained within a stable range relative to input sequence length before and after integrating the Flashback mechanism into Transformer++ and Mamba. Further, the computational cost increase with longer sequences remained within the expected range for subquadratic models, as seen in previous research. Therefore, the proposed method retains robustness even when the input data exceed the learned sequence length (2K tokens).

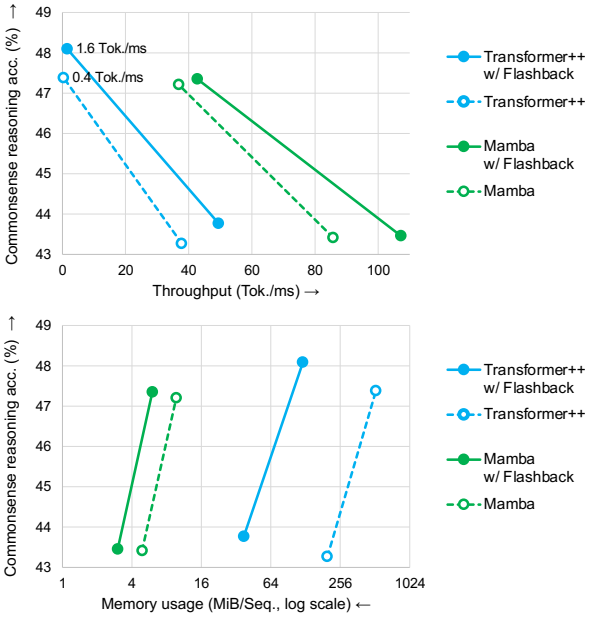

Figure 3: Comparison of the accuracy–performance tradeoffs between the baseline and proposed methods. The accuracy denotes the average performance on commonsense reasoning tasks, similar to that shown in Table 3.

4.5 Ablation Study

4.5.1 Robustness to Hidden State Update Errors

We evaluated the impact of adding noise following the standard normal distribution to the input of blocks during both training and testing. The results are shown in Table 5. Mamba's hidden states depend on all past inputs at each time step, meaning they are affected by noise introduced to past inputs. With the integration of the Flashback mechanism, this dependency is reduced, resulting in improved robustness to noise in the hidden states, as described in Challenges 1 and 2 in Section 1.

4.5.2 Analysis of Hidden State Dependency on Past Inputs

We visualized the frequency histograms of the relative past time indices referenced by the hidden states of each Flashback mechanism in the Mamba-360M model across different training steps (Figure 4). From the figure, we observe that as the model processes more tokens, the distribution of referenced relative time indices becomes progressively narrower. This suggests that the Flashback mechanism selectively retains information over time.

Method	w/ Flashback	Param.	Pile Ppl. ↓	LAMBADA Acc. ↑	HellaSwag Acc. Norm. ↑	PIQA Acc. ↑	ARC-E/-C Acc./Acc. Norm. ↑	WinoGrande Acc. ↑	Average Acc. ↑
H3		360M	10.60	23.58	30.62	63.11	45.20/23.29	50.28	39.35
RWKV v5			9.79	—	—	—	—	—	—
GLA			9.12	—	—	—	—	—	—
BASED			8.65	38.13	33.17	64.58	46.97/24.40	50.59	42.97
Mamba			8.64	39.12	**33.87**	**64.69**	47.85/**24.57**	50.43	43.42
	✓		**8.60**	**40.36**	33.75	64.58	**48.36**/22.35	**51.38**	**43.46**
Transformer++			8.39	38.81	33.61	**64.69**	46.63/24.32	**51.54**	43.27
	✓		**8.38**	**40.89**	**34.13**	64.64	**47.43/25.26**	50.28	**43.77**
Mamba		1.3B	7.48	**46.85**	39.36	**67.57**	51.89/26.11	51.46	47.21
	✓		**7.43**	46.17	**39.46**	67.08	**52.10/26.54**	**52.80**	**47.35**
Transformer++			7.26	48.22	39.08	67.63	51.09/26.11	**52.17**	47.38
	✓		**7.19**	**50.13**	**41.60**	**67.74**	**52.86/26.28**	49.96	**48.09**

Table 3: Comparison of accuracy on The Pile and the commonsense reasoning tasks used in (Gu and Dao, 2023).

Method	w/ Flashback	Param.	Sequence length 2K	8K	16K
Mamba		360M	8.64	8.30	8.26
	✓		**8.60**	**8.24**	**8.20**
Transformer++			8.39	8.38	8.40
	✓		**8.38**	**8.12**	**8.05**

(a) Perplexity ↓ on The Pile dataset.

Method	w/ Flashback	Param.	Sequence length 2K	8K	16K
Mamba		360M	85.7	88.0	88.5
	✓		**107**	**109**	**109**
Transformer++			37.8	22.1	14.1
	✓		**49.4**	**49.6**	**48.3**

(b) Throughput ↑ (Tok./ms).

Method	w/ Flashback	Param.	Sequence length 2K	8K	16K
Mamba		360M	4.9	4.9	4.9
	✓		**3.0**	**3.0**	**3.0**
Transformer++			198	792	1584
	✓		**37.3**	**37.4**	**37.4**

(c) Memory usage ↓ (MiB/Seq.).

Table 4: Comparison of the baseline and proposed methods in terms of accuracy, speed, and memory usage for long-sequence input data.

Method	Param.	w/ Noise Train.	Test.	w/ Flashback	Acc. (%) ↑
Mamba	360M				43.42
				✓	**43.46**
			✓		29.76
			✓	✓	**30.60**
		✓	✓		32.84
		✓	✓	✓	**33.51**

Table 5: Robustness when adding noise to the inputs of blocks. *Acc.* denotes the average performance on commonsense reasoning tasks, similar to that shown in Table 3.

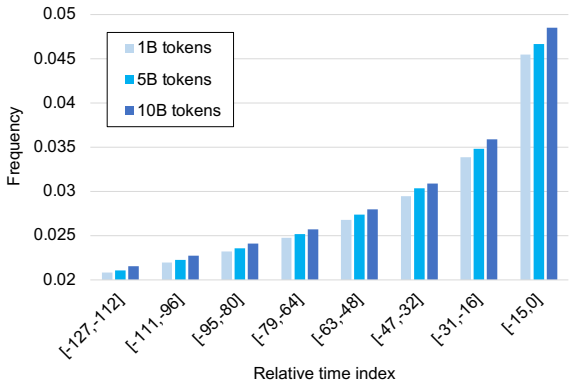

Figure 4: Frequency histogram of the relative time indices referenced by the Flashback mechanism in the final block of the Mamba 360M-based model across different training steps.

4.5.3 Design of Flashback DNN Architecture

We compared commonsense reasoning accuracy as the proportion of the blocks of the Flashback mechanism relative to the total number of blocks in 360M-parameter Mamba varied. The results are shown in Table 6. From the table, we observe that the optimal performance occurs when the number of blocks of the Flashback mechanism and Mamba are balanced at a 1:1 ratio, reflecting a tradeoff between accuracy and processing efficiency.

To prevent the norm of hidden states from increasing as the sequence length grows, we normalized hidden states at each time step (using Norm$(\cdot)$ in Equation (9)). Results showing performance degradation when these this improvement is removed are presented in Table 7. This improvement contributes to the overall performance enhancement of the Flashback mechanism.

4.5.4 Limitations

As shown in Table 8, integrating the Flashback mechanism into Transformer++ and Mamba increases the number of parameters. Thus, although the proposed method improves the tradeoffs between inference accuracy, processing speed, and memory usage, there is a potential risk of overfitting or reduced generalizability due to the parameter increase.

5 Conclusion

We tackled three challenges of previous research on MCNNs: (1) memory degradation, (2) inaccurate

# Mamba blocks	42 (vanilla Mamba)	28	21
# Flashback blocks	0	14	21
Accuracy (%) ↑	42.98	43.50	43.46

Table 6: Accuracy of the Mamba 360M-based Flashback architecture with 42 different numbers of blocks. The accuracy denotes the average performance on commonsense reasoning tasks, similar to that shown in Table 3.

Method	Param.	w/ Hidden state normalization	Sequence length		
			2K	8K	16K
Mamba w/ Flashback	360M	✓	8.67	8.91	10.69
			8.60	**8.24**	**8.20**

Table 7: Reduction in perplexity on the Pile dataset for long-sequence input data achieved through hidden state normalization.

Method	w/ Flashback	Param.	Throughput Tok./ms ↑	Acc. ↑
Mamba		1.3B	36.8	47.21
	✓	1.5B	**42.7**	**47.35**
Transformer++		1.3B	0.38	47.38
	✓	2.2B	**1.57**	**48.09**

Table 8: Impact of introducing the Flashback mechanism on parameter count and performance. *Acc.* denotes the average performance on commonsense reasoning tasks, similar to that shown in Table 3.

gradient backpropagation, and (3) compatibility with next-token prediction. Specifically, we introduced a Flashback property to address (1–2) at a fundamental level and proposed a mechanism that satisfies this property—Flashback mechanism. To address (3), we integrated the Flashback mechanism into Transformers and Mamba in a manner compatible with next-token prediction and demonstrated the effectiveness of the Flashback property through improvements in the tradeoffs between reasoning accuracy, processing speed, and memory usage in experiments. Future work will focus on validating the effectiveness of the proposed method as a memory mechanism for LLMs using larger datasets and architectures.

References

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

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

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2020. PIQA: Reasoning about physical commonsense in natural language. In *AAAI*.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, et al. 2021. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Aydar Bulatov, Yuri Kuratov, and Mikhail S. Burtsev. 2022. Recurrent memory transformer. In *NeurIPS*.

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

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

Alicia Curth, Alan Jeffares, and Mihaela van der Schaar. 2023. A u-turn on double descent: Rethinking parameter counting in statistical learning. In *NeurIPS*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *ACL*.

Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *ICLR*.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *NeurIPS*.

Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, Nanning Zheng, and Furu Wei. 2023. Longnet: Scaling transformers to 1,000,000,000 tokens. *arXiv preprint arXiv:2307.02486*.

Daniel Y. Fu, Tri Dao, Khaled K. Saab, Armin W. Thomas, Atri Rudra, and Christopher Ré. 2023. Hungry Hungry Hippos: Towards language modeling with state space models. In *ICLR*.

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

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, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2024. A framework for few-shot language model evaluation.

Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra. 2023. Imagebind: One embedding space to bind them all. In *CVPR*.

GitHub, Inc. GitHub Copilot.

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

Alex Graves, Greg Wayne, Malcolm Reynolds, et al. 2016. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538:471–476.

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

Albert Gu, Karan Goel, and Christopher Ré. 2022. Efficiently modeling long sequences with structured state spaces. In *ICLR*.

Albert Gu, Isys Johnson, Karan Goel, Khaled Saab, Tri Dao, Atri Rudra, and Christopher Ré. 2021. Combining recurrent, convolutional, and continuous-time models with linear state-space layers. *NeurIPS*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *NeurIPS*.

Neelu Madan, Andreas Moegelmose, Rajat Modi, Yogesh S. Rawat, and Thomas B. Moeslund. 2024. Foundation models for video understanding: A survey.

Wesley J. Maddox, Gregory Benton, and Andrew Gordon Wilson. 2020. Rethinking parameter counting in deep models: Effective dimensionality revisited. *arXiv preprint arXiv:2003.02139*.

OpenAI. ChatGPT.

OpenAI, Josh Achiam, Steven Adler, et al. 2024. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *ACL*.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer. In *ICML*.

Bo Peng, Eric Alcaide, Quentin Anthony, et al. 2023. RWKV: Reinventing RNNs for the transformer era. In *EMNLP*.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. 2023. Measuring and narrowing the compositionality gap in language models. In *EMNLP*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. 2020. Compressive transformers for long-range sequence modelling. In *ICLR*.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature*, 323:533–536.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. WinoGrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*.

Stability AI. Stable Diffusion.

Sainbayar Sukhbaatar, arthur szlam, Jason Weston, and Rob Fergus. 2015. End-to-end memory networks. In *Advances in Neural Information Processing Systems*.

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

Hugo Touvron, Louis Martin, Kevin Stone, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*.

Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. Memory networks. In *ICLR*.

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

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a machine really finish your sentence? In *ACL*.