Language Models Need Sleep

Sangyun Lee[*]
Carnegie Mellon University

Sean McLeish
University of Maryland

Tom Goldstein
University of Maryland

Giulia Fanti
Carnegie Mellon University

Abstract

Transformer-based large language models are increasingly used for long-horizon tasks; however, their attention mechanism scales poorly with context length. To handle this, we study a sleep-like consolidation mechanism in which a model periodically converts recent context into persistent fast weights before clearing its key-value cache. During the sleep, the model performs N offline recurrent passes over the accumulated context and updates the fast weights in its state-space model (SSM) blocks through a learned local rule. During inference, this shifts extra computation to the sleep while preserving the latency of wake-time prediction. We test our method on controlled synthetic tasks, including cellular automata and multi-hop graph retrieval, as well as a realistic math reasoning task, on which a regular transformer as well as SSM-attention hybrid models fail. We then show that increasing sleep duration N for our models improves performance, with the largest gains on examples that require deeper reasoning.

1 Introduction

Large Language Models (LLMs) are commonly based on the transformer architecture [51], which stores context in an attention cache and retrieves past tokens as needed. This memory mechanism is central to their performance, but it scales poorly: total attention compute grows quadratically with context length, while cache memory grows linearly.

Recent efficient sequence models [42, 18, 16, 2] mitigate this cost by introducing fixed-size fast weight memories [54, 15, 43] interleaved with full self-attention. This hybrid design brings together two complementary forms of memory: attention for high-fidelity access to recent tokens, and weight-based memory for compressed information beyond the active context window. Hybrid models are now common among large scale frontier models [49].

However, scalable memory is not the same as scalable reasoning. A fast weight memory may support long-range recall [42], but it is unclear whether it can support deep computation over tokens that are no longer present in the KV cache. We find that the performance of vanilla SSM-attention hybrid models degrades (under the same token budget) as the required reasoning depth increases **even when the amount of information to store is held fixed.** This suggests that the bottleneck is not merely memory capacity as suggested by prior work [27, 2], but the amount of computation available for transforming evicted context into a useful internal state.

Sleep. In animals, the transfer from short-term memory to long-term memory is thought to be supported by hippocampal replay [33], especially during sleep [41]; in this phase, short-term hippocampal memories are reactivated and consolidated into cortical synaptic weights. Sleep makes animals unable to respond to external stimuli, suggesting that it must provide enough cognitive benefit

[*]Correspondence to: sangyunl@andrew.cmu.edu

to justify this cost [41]. Inspired by these biological processes, we propose a method for transferring context-window memory into persistent weights. When the model's context window becomes full during inference, the model enters a "sleep" in which it performs multiple forward passes over the accumulated context and recursively updates its fast weights via a learned local rule. As in animal sleep, the model receives no external input tokens during this phase. After consolidation, the context window is cleared, and the model resumes operation with updated fast weights. During training, the model is optimized end-to-end by backpropagating through the entire process to maximize task performance after sleep.

Our architecture is also motivated by results on depth-recurrent or looped neural networks [23, 17, 4]. Prior work shows that dynamic-depth models can outperform fixed-depth counterparts on sequential reasoning tasks and solve hard problem instances that fixed-depth models cannot by scaling amount of compute spent at prediction. **Our key insight is that recurrence can be used not only for prediction but also for memory consolidation.** Converting observed tokens into useful weight memory is itself a nontrivial computation, and need not be achievable in a single pass. Indeed, many learning algorithms, such as gradient descent, improve through iterative weight updates. Thus, allocating more recurrent computation during fast weight formation gives the model more steps to transform context into representations that support later prediction. We find that increasing the depth of recurrence, or *sleep duration*, improves reasoning after sleep. Unlike previous looped models, our model does not need to loop at prediction time: the additional computation has already been spent on forming fast weights that support later single-pass prediction.

We introduce and evaluate LLM sleep on carefully designed *synthetic tasks* where a model must answer questions about context that has already been evicted, using only a single forward pass. These synthetic tasks allow us to vary reasoning depth while holding memory load fixed, providing a clean stress test of whether sleep-time computation can convert transient context into fast weights that support later inference. We summarize our contributions as follows:

- In a controlled setting, we show that as the reasoning depth of a problem increases, vanilla State-Space Models (SSMs) such as Gated Delta Nets (GDNs) fail despite having enough fast weight capacity.
- We propose an architecture that combines recurrent computation with fast weight memory blocks, and show that increasing the number of recursions for our architecture improves performance over GDNs. We observe the largest gains on problem instances that require the deepest reasoning.
- We further validate the efficacy of our architecture on GSM-Infinite, a natural language math-reasoning dataset, using pre-trained LLM initializations.

Overall, these results support the central claim that a sleep-like offline recurrence can organize evicted context into weights to support later reasoning.

2 Related Work

Fast weights and linear recurrent neural networks. Linear recurrent neural networks or SSMs can be viewed as maintaining an online fast weight memory rather than a KV cache which grows quadratically with sequence length. In this view, linear attention corresponds to a recurrent update over a fixed-size, matrix-valued, state, where key-value mappings are written and queried [29, 43]. Recent variants improve this memory with delta-rule updates and gates, enabling more selective writing, overwriting, and forgetting [53–55, 15]. These mechanisms underlie recent efficient hybrid language models [24, 39] and help explain why linear networks can offer a favorable recall, throughput, and memory tradeoffs. They still struggle with exact copying and retrieval relative to full attention in some cases due to a fixed memory size, as pointed out by prior work [2, 27]. Contrary to these works, we show that such models can fail as the required reasoning depth to solve a task increases, *even when the amount of information to store is held fixed*.

Context compression. There are several methods for processing long contexts at test time by condensing contextual information. Ge et al. [21] propose using a language model to compress long contexts into a shorter sequence of hidden states, which are then passed to the language model in place of the original long context. Eyuboglu et al. [20] use offline self-study to learn a small KV cache that can substitute for the full-context cache. This line of work shares our goal of spending offline computation once to turn a long context into a compact state that can be reused later. These

methods shorten what remains in the attention context, whereas our method transfers evicted context into weight-based memory.

Context distillation. Context distillation [46, 3] aims to distill active context into model weights by training a model without it to imitate a contextful teacher [46, 3, 8], reconstruct it [11], predict its continuation [8, 11], or answer questions about it [47, 9, 8]. Instead of doing gradient descent on predefined losses, our method uses a learned recurrent forward pass to transfer context to weights.

Test-time training. Tandon et al. [48] replace full attention with sliding-window attention and perform test-time gradient updates on a subset of MLP layers. At inference time, their method optimizes a standard cross-entropy loss on the observed context, storing long-range information in temporary parameter updates rather than in a full KV cache. They perform only one gradient step for distilling each context chunk. By contrast, our method uses a learned recurrent forward pass as the memory-update rule, allowing more flexible forms of consolidation that need not correspond to a one-step gradient descent on a fixed scalar objective. They primarily evaluate perplexity on general web-text data, where retrieval and reasoning demands are entangled; we instead use synthetic tasks that independently control reasoning depth and problem length, showing that additional sleep-time computation is most beneficial when reasoning depth increases. Zhang et al. [56] attach a LoRA adapter that updates model weights from the current context chunk and evaluate this approach in a reinforcement-learning setting. Unlike ours, their method updates the weights only once per chunk.

Depth-recurrent models. Increasing the depth of language models is known to increase their expressivity [35]. Depth-recurrence, is one way to increase depth in transformer models and is one method to make them Turing complete [17]. Moreover, the depth of these models can be adaptive [23, 19, 44, 5]. Recent work has scaled these depth-adaptive language models to large scales, both training from scratch [22, 58] and as a post-training objective [34]. Detailed analyses of how best to train depth recurrent models suggest the recurrent depth should be scaled with training compute [40, 45].

Offline planning. Successful planning in structured environments often requires combining newly-observed information with memories of earlier states. A longstanding view is that animals perform this integration online at choice time [50, 36]. However, integrating distant memories at choice time can be time-consuming, and offline planning during off-task rest can amortize such cost [36]. Consistent with this view, Momennejad et al. [36] show that neural evidence of offline replay during rest predicts improved planning performance for human subjects. Recent work from the machine learning community studies related mechanisms with artificial neural networks. Lin et al. [30] propose scaling offline compute by letting LLMs generate expected questions from users and precompute quantities needed to solve them. Chalvidal et al. [10] train a single-layer network on reinforcement-learning environments and show that recursive Hebbian-like weight updates support fast adaptation. In this paper, we show that recursively updating fast weights during a sleep-like offline phase improves reasoning over evicted context while preserving a strict prediction-phase latency constraint.

3 Preliminaries

3.1 Sequence mixers

Attention. Softmax attention [51] is a sequence-mixing operation in which each token retrieves information from previous tokens according to query-key similarity. For the token representation $\boldsymbol{x}_t$ at timestep t, define

$$\boldsymbol{q}_t = \mathbf{W}_Q \boldsymbol{x}_t, \qquad \boldsymbol{k}_t = \mathbf{W}_K \boldsymbol{x}_t, \qquad \boldsymbol{v}_t = \mathbf{W}_V \boldsymbol{x}_t, \tag{1}$$

where $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t \in \mathbb{R}^d$ are column vectors, and $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$ are learned projection matrices with compatible shapes. Self-attention stores all previous keys $\boldsymbol{k}_t$ and values $\boldsymbol{v}_t$ in $\mathbf{K}_t = [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_t]^\top \in \mathbb{R}^{t \times d}$ and $\mathbf{V}_t = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_t]^\top \in \mathbb{R}^{t \times d}$, then computes

$$\boldsymbol{o}_t = \mathbf{V}_t^\top \operatorname{softmax}\left(\frac{\mathbf{K}_t \boldsymbol{q}_t}{\sqrt{d}}\right). \tag{2}$$

This allows $\boldsymbol{x}_t$ to attend to any previous token, but requires storing $\mathbf{K}_t$ and $\mathbf{V}_t$, the KV cache, whose size grows linearly with sequence length.

Linear recurrent layers. By contrast, linear recurrent layers, including many SSM-style architectures, store the past in a fixed-size fast-weight state. A simple Mamba2-style [15] update can be written as a gated Hebbian-like outer-product rule [25, 43]:

$$\mathbf{S}_t = \alpha_t \mathbf{S}_{t-1} + \beta_t \boldsymbol{v}_t \boldsymbol{k}_t^\top, \qquad \boldsymbol{o}_t = \mathbf{S}_t \boldsymbol{q}_t. \tag{3}$$

Here $\alpha_t \in (0,1)$ is a data-dependent forget gate and $\beta_t \in (0,1)$ is a data-dependent input gate, both computed from $\boldsymbol{x}_t$. Unlike the KV cache $\mathbf{K}_t$ and $\mathbf{V}_t$, the fast-weight $\mathbf{S}_t$ does not grow in size with t. This makes linear recurrent layers more memory-efficient, but also more lossy: past tokens must be compressed into a fixed-size weight-based memory. In our experiments we use Gated Delta Networks (GDNs), which add a delta-rule correction to this update; however, the specific update rule does not matter for our discussion.

In a language model, a sequence-mixing layer is combined with normalization, residual connections, and an MLP layer to form a block. We write $\mathcal{B}_\ell^{\text{attn}}$ for a block whose sequence-mixing layer is attention, and $\mathcal{B}_\ell^{\text{ssm}}$ for a block whose sequence-mixing layer is a linear recurrent layer.

For example, an attention-only language model is formed by stacking attention blocks D times between an embedding layer and an output projection:

$$\text{Embed} \to \mathcal{B}_0^{\text{attn}} \to \cdots \to \mathcal{B}_\ell^{\text{attn}} \to \mathcal{B}_{\ell+1}^{\text{attn}} \to \cdots \to \mathcal{B}_{D-1}^{\text{attn}} \to \text{OutProj}. \tag{4}$$

Hybrid models. Recent hybrid sequence models [42, 18, 16, 2] mitigate the cost of self-attention layers by interleaving them with SSM blocks [54, 15, 43] with fixed-size fast-weight memories. For example:

$$\text{Embed} \to \mathcal{B}_0^{\text{attn}} \to \mathcal{B}_1^{\text{ssm}} \to \mathcal{B}_2^{\text{attn}} \to \mathcal{B}_3^{\text{ssm}} \to \cdots \to \mathcal{B}_{D-1}^{\text{attn}} \to \text{OutProj}. \tag{5}$$

3.2 Synthetic reasoning tasks

To begin, we study two synthetic tasks to understand our changes in a controlled setting.

Rule 110. Rule 110 [13] is a simple one-dimensional binary cellular automaton that evolves a binary string according to a fixed local transition rule. The general problem of predicting Rule 110 after t steps is P-complete [37], and no efficient general parallel shortcut is known. Training a neural network to predict the t-th state is therefore a good test to see if the model can carry out deep sequential computation.

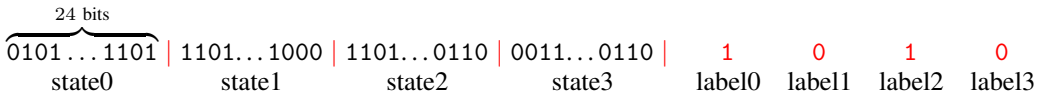
Depo. Depo is a multi-hop knowledge retrieval task introduced by Allen-Zhu and Li [1] to evaluate reasoning depth of a language model. Each sequence consists of a shuffled directed cycle followed by queries; each query asks for the node reached after k outgoing edges from a start node, with larger k requiring deeper graph traversal.

These tasks allow us to vary reasoning demand while holding sequence length fixed, isolating a model's reasoning capability from its information retrieval capability.

4 Motivating example: Can attention-SSM hybrid models reason about context they can no longer attend to?

Attention-SSM hybrid models are often motivated by the idea that fast-weight memory can compensate for limited attention windows [42], compressing information from past tokens once they are no longer directly accessible. In this section, we explore a case where this hybrid mechanism fails.

Consider the following example drawing on cellular automaton Rule 110 [13]. In this setting, we train the model on four independent length-24 binary strings, each representing an initial state for Rule 110. Here, we use a character-level tokenizer (i.e., '0' and '1' define tokens). The four states are unrelated to each other (i.e., they are not obtained by unrolling the previous state). After processing the all four binary strings of length $T := 24 \times 4 = 96$, the model must later predict the first bit of each state after t transitions. Since there are four label tokens following the states, the total sequence length T is 100. An example sequence is:



The first answer token 1 (label0) is obtained by unrolling 0101...1101 (state0) t times and taking the first bit from it, and so on. t controls the reasoning depth required to solve this task: when $t = 0$ (no rollout), this becomes a simple first-bit retrieval task, and the task becomes more difficult as t increases.

To stress-test whether SSM can complement self-attention by providing past information, we impose a strict context window size as well as a *hard-eviction constraint*: we clear the context window every 24 tokens, and we denote this with $L = 24$. This means that the model can only see one state in context at a time and must fully encode this information into its fast weights $\mathbf{S}_t$, as the KV cache $\mathbf{K}_t$ and $\mathbf{V}_t$ are *fully evicted* before moving onto the next state. The hard eviction boundary is denoted by |.

This hard eviction constraint naturally divides a sequence into two distinct phases:

- the **consolidation phase** (the first 96 tokens in the example sequence), during which the model must encode context into its fast weights $\mathbf{S}_t$; and
- the **prediction phase** (the last 4 tokens in the example sequence), during which the model predicts the answer tokens.

We impose a *prediction-phase latency constraint*: during the prediction phase, each answer token is predicted with a single standard forward pass. Extra loops or chain-of-thought tokens are disallowed because they increase prediction latency. Thus, all information needed to predict the labels must already have been consolidated into the fast weights before the prediction phase begins.

Under this hard eviction constraint, a standard transformer cannot do better than random guessing as the KV cache has been destroyed before prediction is made. SSMs or attention-SSM hybrid models can do better than random guessing because they can store the initial states in their fast weights. For example, one way to solve this task is to simulate the t-step state evolution once the context is full, store the first bit of each evolved state in the fast weights, and retrieve this bit at prediction time. However, Figure 2a shows that the performance of a 4-layer GDN-attention hybrid model (with an attention $\rightarrow$ GDN $\rightarrow$ attention $\rightarrow$ GDN layout) drops rapidly as t increases. This drop is not due to the memory-capacity limitation identified in prior work [27, 2]: we vary only t while keeping the sequence length T fixed. Instead, the difficulty comes from the deep sequential computation needed to simulate the automaton for t steps, which a fixed-depth model cannot scale with.

On task failures. When we say that a model fails or degrades on a task, we do not mean that the architecture could never learn the task with unlimited data, compute, or training time. Our claims concern performance under a fixed training-token budget. This budgeted setting matters because reasoning-intensive data is sparse even in web-scale corpora. Budget-controlled synthetic tasks can expose trends that align with phenomena observed in larger-scale pretraining earlier and more clearly [1].

5 LLM Sleep: Offline Recursive Memory Consolidation

Now, we introduce a solution to the above example: we introduce a *sleep* during LLM training, in which the model performs recursion during a consolidation phase, before evicting tokens from attention layers once the context window is full. In this way, we can scale compute to handle deep reasoning tasks (e.g., a large t from our motivating example) while still obeying a prediction-phase latency constraint. For example, if we loop over all D blocks, it looks like:

$$\text{Embed} \rightarrow \left[\mathcal{B}_0^{\text{attn}} \rightarrow \mathcal{B}_1^{\text{ssm}} \rightarrow \cdots \rightarrow \mathcal{B}_{D-1}^{\text{attn}}\right]^{\times N} \rightarrow \text{OutProj} \tag{6}$$

where the superscript $\times N$ denotes N looped passes over the architecture.

Figure 1 describes the architecture in detail. We initialize from an SSM-attention hybrid model with a fixed context-window size L, where the attention cache is fully evicted every L tokens. Before evicting the KV cache every L tokens, the model performs N recurrent passes to iteratively update the fast weights inside the SSM blocks following Equation (3); with $N = 1$, it reduces to a vanilla SSM-attention hybrid model. We call the phase when the model is iteratively updating the fast weights a **sleep**.

Algorithm 1 Our LLM sleep training with hard eviction.

Require: tokens x, loss mask m, window size L, sleep passes N
Zero-initialize SSM fast weights $\mathbf{S}$
Split x, m into non-overlapping chunks of length at most L
for each token chunk c and its loss mask m_c **do**
  $h \leftarrow \mathrm{Embed}(c)$
  if m_c is all-zero **then** ▷ consolidation phase
    for $n = 1, \ldots, N$ **do**
      $h, \mathbf{S} \leftarrow \mathrm{Blocks}(h, \mathbf{S})$
    end for
  else ▷ prediction phase
    $h, \mathbf{S} \leftarrow \mathrm{Blocks}(h, \mathbf{S})$
    $\mathcal{L} \leftarrow \mathrm{MaskedCE}(\mathrm{OutProj}(h), c, m_c)$ ▷ Masked cross entropy loss
  end if
end for
Backpropagate $\mathcal{L}$ and take an optimizer step

After recurrently refining the fast weights, the KV cache is evicted and the next L tokens are processed. After processing the full context, the model predicts the answer based on the refined memory and current context *in a single forward pass*. The model is trained to minimize the prediction error by backpropagating through the entire computational graph shown in Equation (6), similarly to other depth-recurrent models [17, 23]. Unlike prior depth-recurrent models where gradient flows through recursively refined feature vectors, the gradient flows through the refined fast weights because we discard the refined features after sleep. Algorithm 1 summarizes the training procedure.

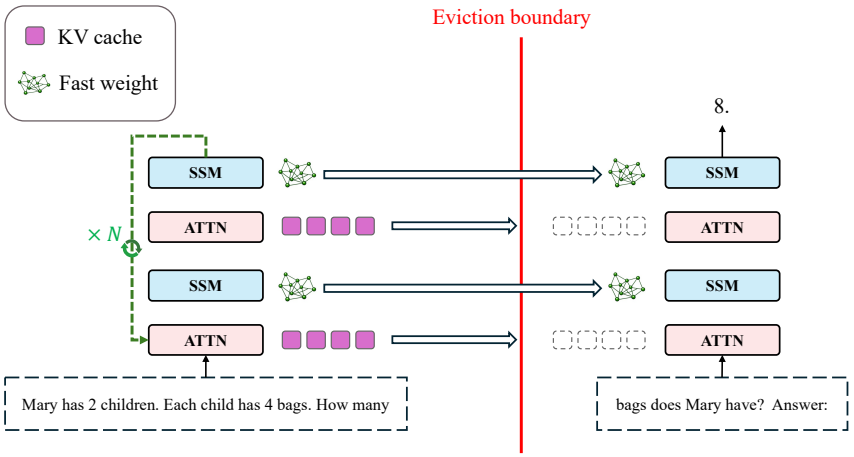
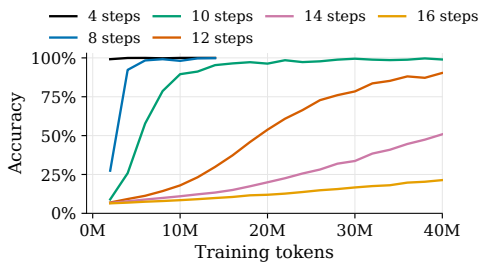


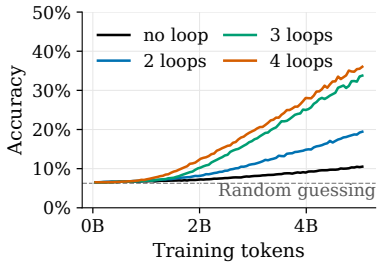
Figure 1: At the eviction boundary, an SSM-attention hybrid performs N offline recurrent passes over the current context before discarding the attention cache. These recurrent passes update the fast weights in the SSM blocks, allowing later predictions to use consolidated context without wake-time looping.

6 Experiments

Our experiments test whether longer sleep, implemented by increasing N, produces fast weights that support deeper reasoning over states that are no longer present in the attention cache. This requires more than storing evicted tokens: the model must encode past context into fast weights $(\mathbf{S}_t)$ in a form that supports nontrivial computation after the cache has been cleared, while still using only a single forward pass at prediction time. We evaluate this question across increasingly more difficult settings. First, the cellular automaton task varies the rollout step t, isolating the depth of reasoning required over each evicted state. First, the Depo task [1] adds a harder compression problem: the model must encode a fragmented graph into fast weights and later answer unseen multi-hop queries



(a) Effect of rollout step t.



(b) Effect of offline looping for $t = 32$.

Figure 2: **Increasing N improves performance on cellular automaton. Left:** Each curve represents a different number of rollout steps t for a hybrid attention-SSM architecture, as in the motivating example section. Increasing t makes the task harder for a vanilla attention-GDN hybrid model. We early-stop 4- and 8-step runs as they converge earlier. **Right:** For a challenging reasoning task ($t = 32$), additional offline sleep loops improve accuracy while preserving single-pass wake-time prediction.

over it. Finally, we consider GSM-Infinite [57], where we fine-tune the pre-trained Jet-Nemotron 2B [24] and Ouro 1.4B [58] on a synthetic math-reasoning dataset.

Experiment details. Following McLeish et al. [34], we use the Muon optimizer for all experiments. We fix the AdamW learning rate to $5\text{e}{-5}$ and tune only the Muon learning rate. For Section 4 and Section 6.1, we use a 4-layer GDN-attention hybrid model with hidden dimension $d = 256$. We tune the Muon learning rate on the $N = 1$ model, giving the no-loop baseline an advantage, and use the selected value, $2\text{e}{-3}$, for all looped models. For Section 6.2, we use the Jet-Nemotron architecture [24], an SSM-attention hybrid model fine-tuned from Qwen 2.5 1.5B by replacing some attention layers with Jet layers, which use dynamic convolution instead of the fixed convolution in GDN. To roughly match the small model size in Allen-Zhu and Li [1], we train a 10-layer model from scratch with hidden dimension $d = 512$. We apply the same tuning protocol as above: tune the Muon learning rate on the $N = 1$ baseline and use $2\text{e}{-3}$ for the looped models. For Section 6.3, we use pre-trained Ouro 1.4B [58] and Jet-Nemotron 2B [24] models, and set the Muon learning rate to $1\text{e}{-3}$ following McLeish et al. [34]. The automaton experiments require less than one A6000 GPU-day. The Depo and GSM-Infinite experiments require roughly 1–2 H100 GPU-days per run. For the batch size, we use 512 for automaton, 128 for Depo, and 256 for GSM-Infinite. For fair comparison, we fix random seeds ensuring that all runs use exactly the same data ordering.

6.1 Task: Cellular automaton

In Section 4, we see how vanilla SSM-attention hybrid models fail on the automaton task when t is large, as hybrid models cannot scale compute when performing memory consolidation. In Figure 2b, we use the same architecture from Figure 2a: a 4-layer GDN-attention hybrid model, with an attention $\rightarrow$ GDN $\rightarrow$ attention $\rightarrow$ GDN layout. Our method additionally uses the 'sleep' during the consolidation phase discussed in Section 5, where we use recurrence to iteratively update the fast weights. We study using 2 to 4 recurrent updates here.

We train this looped hybrid architecture on a setting that requires substantial reasoning compute and is challenging for the non-recurrent architecture: $t = 32$. In Figure 2b, "2 loops", "3 loops", and "4 loop" mean the model uses a *sleep* for memory consolidation, while "no loop" is the baseline. Figure 2b shows that the non-looped model remains close to random guessing, reaching only about 10% exact accuracy after nearly 5B training tokens. Adding offline passes improves both learning speed and final accuracy under the same token budget: two loops achieves approximately 20% accuracy, while three and four loops achieve above 30%. Because the context length, eviction rule, and prediction-phase computation are fixed across these runs, the improvement comes from additional consolidation-time computation during sleep.

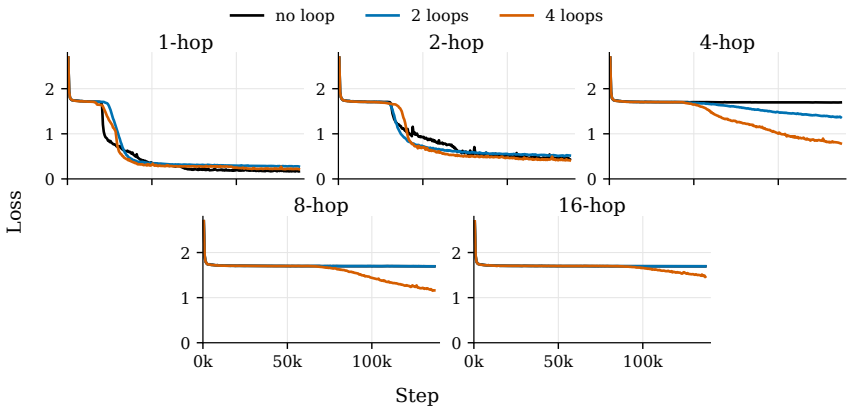


Figure 3: **Increasing N improves performance on Depo.** Test loss of a 4-layer GDN-attention hybrid on the k-hop knowledge retrieval task. Additional offline loops accelerate learning, especially for more reasoning-intensive, higher-hop queries.

6.2 Task: Depo

Next, we evaluate Depo, the k-hop knowledge retrieval task introduced by Allen-Zhu and Li [1]. Each sequence consists of a shuffled directed cycle followed by queries; each query asks for the node reached after k outgoing edges from a start node, with larger k requiring deeper graph traversal. Allen-Zhu and Li [1] show that SSMs perform substantially worse than transformers on this task, despite having enough fast weight capacity to store the context. This suggests that the bottleneck is not storage alone, but organizing stored edges into a representation that supports later multi-hop retrieval [38]. An example sequence from Depo is:

shuffled directed cycle: `b->a, f->l, ... | ... | ..., e->b`

query and answer: `1 hop after a: c ... 4 hops after e: d`

Here | denotes an eviction boundary, and red text denotes answer tokens.

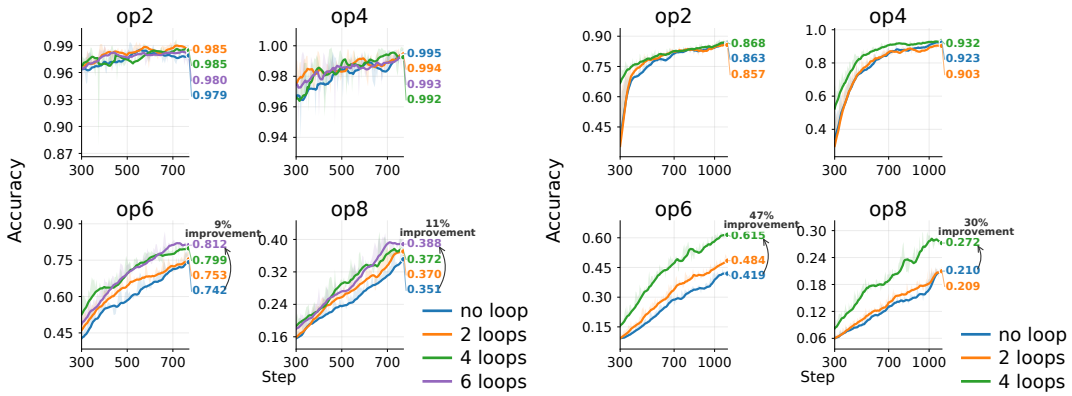
In our setting, each cycle contains up to 75 nodes and spans up to 300 tokens; shorter instances are left-padded to 300 tokens both at test and train time. The query-answer portion then follows, with 10 query-answer pairs spanning up to 60 tokens, making the total sequence length $T = 360$. The model's window size is $L = 75$, so each cycle is fragmented across four cache windows. When the model predicts the query answers, the cycle context has been evicted from the KV cache. Depo is harder than the cellular automaton task for two reasons. First, each cycle is fragmented across four cache windows, whereas each automaton state fits within a single window. Second, the model must form a query-agnostic representation because both k and the start node are randomly sampled for each example, whereas t is fixed in the automaton task.

In Depo, k controls task difficulty: larger k makes the query more difficult because the model must perform longer multi-hop traversal to recover the answer. Following Allen-Zhu and Li [1], we uniformly sample k from $[1, 16]$ during training and measure test loss on held-out examples with $k = \{1, 2, 4, 8, 16\}$.

Figure 3 shows test loss on held-out examples over training steps, with each subplot corresponding to a hop count $k \in \{1, 2, 4, 8, 16\}$ and each curve comparing a model with $N \in \{1, 2, 4\}$ offline loops. We see that increasing the number of offline loops improves learning speed for queries that require 4 or more hops. The 1-loop model makes little progress on 4-hop and harder queries, and the 2-loop model similarly stalls on 8-hop and harder queries. Within our training budget, only the 4-loop model begins to improve on the hardest 16-hop task.

6.3 Task: GSM-Infinite

To test whether the trend from the controlled tasks extends to pretrained LLMs, we evaluate on GSM-Infinite [57], a synthetic reasoning benchmark modeled after GSM8K [12]. GSM-Infinite is



(a) Jet-Nemotron 2B.

(b) Ouro 1.4B.

Figure 4: **Increasing N improves performance on GSM-Infinite.** GSM-Infinite accuracy over training steps. Subplots group examples by the number of arithmetic operations required by the problem, and colors indicate the number of offline loops N used before cache eviction. Additional loops improve accuracy most clearly on harder problems with more operations, where single-loop models have less sleep-time computation available to organize the evicted context into useful fast weights.

still structured enough for controlled analysis, but realistic enough that training on it can improve a model's reasoning capabilities on other tasks [28]. As GSM-Infintie is procedurally generated we can generate distinct training and evaluation datasets from the same distribution, similarly to Kabra et al. [28]. Our evaluation set is 1,600 held-out examples. The dataset controls problem length by adding distractor tokens that resemble the rest of the problem, making them difficult to ignore, and controls difficulty by varying the number of arithmetic operations required to solve the problem. Unlike retrieval-focused long-context tasks such as RULER [26], simple retrieval-augmented baselines fail [57] on GSM-Infinite, indicating that the task requires both long-context processing *and* multi-step reasoning. GSM-Infinite is challenging even for reasoning-optimized frontier models, whose accuracy decays as the number of required operations increases [57].

In our experiments, each problem contains between 2,000 and 3,300 tokens, and the number of operations is sampled uniformly from $[1, 8]$. We place the question before the context and exclude Chain-of-Thought traces from the data, forcing the model to the final answer in the single prediction time forward pass alone. This order gives the model the query before it reads the long problem context, allowing it to selectively consolidate information relevant to the question while ignoring filler tokens. We set the model's context-window size to $L = 2000$, so a full problem does not fit in the active context window and the model cannot attend to a majority of the problem context at prediction time.

There are two complementary ways of instantiating our method from a pre-trained model: starting from an SSM-attention hybrid and fine-tuning it with sleep time recurrence, or starting from a depth-recurrent model and adding SSM memory layers. We explore both, fine-tuning the hybrid Jet-Nemotron 2B [24], and the recurrent Ouro 1.4B [58]. Jet-Nemotron is an SSM-attention hybrid model fine-tuned from Qwen 2.5 1.5B by replacing some attention layers with Jet layers, which use dynamic convolution instead of the fixed convolution in GDN. Ouro is a looped attention-only model, so we insert 6 Jet layers without MLP layers to augment Ouro with fast weight memory while increasing the total parameter count by less than 10%.

For Jet, we loop over the middle 14 blocks out of the total 28 blocks. Looping over middle-blocks only is a common practice in depth-recurrence models [34, 22]. For Ouro, we loop over the entire blocks following how the model is pre-trained [58]. To keep memory cost during training manageable while using a reasonable batch size, we use $N = \{1, 2, 4\}$ for Ouro. Since Jet loops over only a half of the entire blocks, we use $\{1, 2, 4, 6\}$.

Figure 4 shows the accuracy trend over training steps, with each subplot corresponding to a different number of operations required to solve the problem, ranging from 2 to 8. We see that the trend from

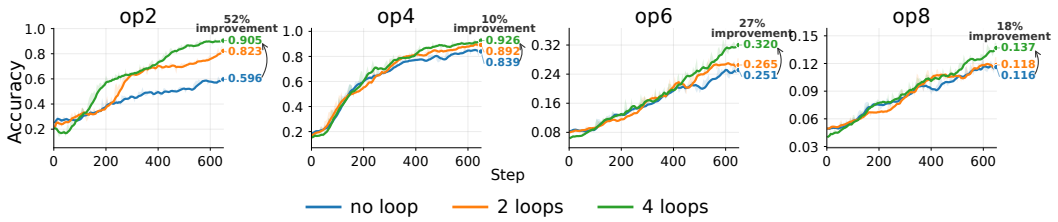


Figure 5: **Increasing N improves accuracy on GSM-Infinite with sliding-window eviction.** GSM-Infinite accuracy with sliding-window eviction over training steps. We fine-tune Ouro 1.4B with window size $L = 512$ and compare $N \in \{1, 2, 4\}$ sleep passes.

the pretraining from scratch experiments persists in a more realistic math-reasoning setting. For easier two- and four-operation problems, accuracy often approaches saturation regardless of the number of loops, especially for Jet, which has more fast weight memory capacity than Ouro. However, as the number of required operations increases, the gap between loop counts widens: additional offline recurrence improves both final accuracy and learning speed on the six- and eight-operation settings. For Jet, six loops improves final accuracy on six-operation problems from 0.742 to 0.812 and on eight-operation problems from 0.351 to 0.388. For Ouro, four loops improves final accuracy from 0.419 to 0.615 on six-operation problems and from 0.210 to 0.272 on eight-operation problems. The gap is wider for Ouro, which may reflect its depth-recurrent pretraining. These results suggest that sleep-time computation can support multi-step reasoning even on realistic math-reasoning data and with pre-trained LLMs.

6.4 Sliding-window eviction

So far, we have assumed that the model's context window is completely evicted whenever it is full. We can instead use a sliding-window eviction strategy: after sleep, the model retains the most recent $L - 1$ tokens in the attention cache and evicts only older tokens. This does not increase peak inference-time memory: the active context is still capped at L tokens, as in sliding-window attention (SWA). With $N = 1$, this reduces to a standard SWA-SSM hybrid model [42]; with $N > 1$, the model performs additional recursive consolidation before older context leaves the attention cache.

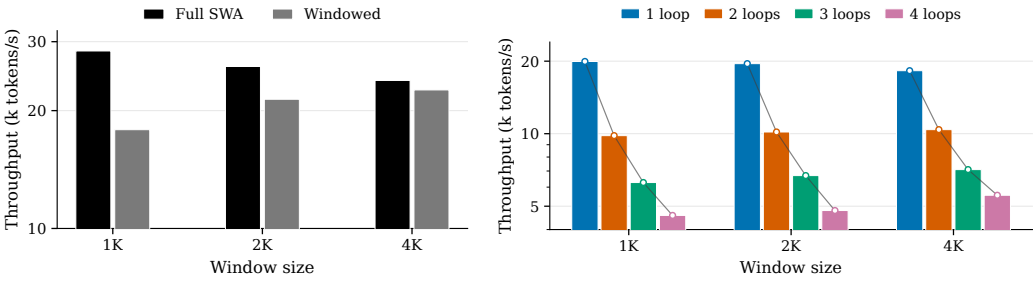
We evaluate this strategy on GSM-Infinite with $L = 512$, so the total sequence length T is roughly 4–6$\times$ the window size. We fine-tune Ouro 1.4B with $N \in \{1, 2, 4\}$. Analogously to observations in prior work [7], we find that giving the model access to a sliding-window KV cache can make the newly inserted Jet layers underutilized. We therefore first warm up only the Jet layers for one epoch and then train the full model for two epochs. This SSM-only warm-up stage is standard when converting attention-only models into attention-SSM hybrids [52, 6, 24]. We find that for $N > 1$, using hard eviction for the warm-up stage is crucial for the model to learn to refine the fast weights.

Figure 5 shows accuracy over training steps, where the curve labeled `no loop` corresponds to the SWA-SSM hybrid baseline with $N = 1$. Increasing N improves accuracy at all operation counts, matching the trend in Figure 4. Unlike in Figure 4, where the window size is $L = 2000$, this baseline performs poorly even on two-operation problems, which are the least reasoning-heavy and therefore more directly stress retrieval under distractor tokens. On the other hand, using loops drastically improves accuracy from 0.596 to 0.905, an 52% improvement. This suggests that when the active attention window is several times smaller than the sequence length, **longer sleep duration helps not only with multi-step reasoning, but also with compressing and retrieving relevant context.**

6.5 Training throughput

Here we analyze how our method affects training throughput in terms of the number of tokens processed per second compared to a SWA-SSM hybrid baseline. We use Ouro 1.4B model from Section 6.3.

Recurrence across context windows. Unlike standard teacher-forced transformer training, which can process all token positions in parallel, our training is recurrent across context windows, since before window $j + 1$ can be processed, the model must finish processing window j and perform



(a) Parallel SWA vs windowed.

(b) Effect of varying N.

Figure 6: **Recurrence across context windows incur minimal training overhead; recurrent-depth linearly increases cost.** Training throughput comparison on 1 NVIDIA H200 GPU. Sequence length is set to $12,000$. (a) When window size L is sufficiently large, serialness across context windows do not meaningfully change the throughput compared to the fully parallel baseline. (b) Throughput is roughly inversely proportional to N. For each setting, batch size is tuned to optimize the GPU utilization. (b) additionally uses activation checkpointing across context chunk axis to prevent out-of-memory error. FlashAttention 2 [14] is used.

the N sleep passes that refine the fast weights. The updated fast weights then become the state used to process window $j+1$, creating a sequential dependency across windows. This prevents full parallelization along the sequence axis. However, this loss of sequence-axis parallelism need not hinder wall-clock training time when the window size L is large enough to keep the GPU saturated, as can occur in long-context training regimes where both T and L are large, as shown in Figure 6a.

Recurrent-depth cost. In addition, as in other depth-recurrent models, training cost grows roughly linearly with the number of recurrent steps N, as shown in Figure 6b. However, as we see in our experiments, increasing recurrence consistently improves task performance compared to non-recurrent models.

7 Discussion and Limitations

Our method preserves single-pass prediction-phase latency by moving the extra recurrent computation into the consolidation phase, but this gain is not free: during training, we need to perform N deeper forward and backward passes, which can make training slow and unstable. Tackling these challenges is an active topic in recurrent-depth training, with possible approaches including implicit gradients [4] and truncated backpropagation through time [22, 34], as well as various techniques to stabilize training [40, 22].

Sleep makes training sequential across context and depth dimension, but this sequentiality is also why our method shows gains on the tasks we consider, whose solutions are themselves sequential. Many reasoning, simulation, and decision-making problems often targeted by modern machine learning appear to have this property [32]. Attempting to solve inherently sequential tasks with fully parallel computation encourages brittle shortcut solutions [31, 32].

8 Conclusion

We propose a sleep-like process in which a model performs multiple recursive forward passes to iteratively refine its fast weights before evicting the corresponding context from the attention cache. Unlike vanilla attention-SSM hybrid model, sleep allows models to reason deeply about past context that they can no longer attend to. Across controlled synthetic tasks and a more realistic mathematical reasoning benchmark, we show that increasing the number of recursions, or sleep duration, improves the model's ability to perform deep sequential computation over evicted context.

Broader Impact

This work studies memory consolidation and reasoning in language models, which are important ingredients for building more capable long-context systems. Our contribution is primarily methodological and is evaluated on controlled synthetic tasks and modest-scale pretrained models. We therefore do not expect the risks to exceed those of other work in this area.

Acknowledgements

We gratefully acknowledge Modal for providing generous GPU resources.

References

[1] Zeyuan Allen-Zhu and Yuanzhi Li. Physics of language models: Part 4.1, architecture design and the magic of canon layers. In *The Thirty-Ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=kxv0M6I7Ud.

[2] Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff. *arXiv preprint arXiv:2402.18668*, 2024.

[3] Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Jackson Kernion, Kamal Ndousse, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, and Jared Kaplan. A general language assistant as a laboratory for alignment, 2021. URL https://arxiv.org/abs/2112.00861.

[4] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in neural information processing systems*, 32, 2019.

[5] Arpit Bansal, Avi Schwarzschild, Eitan Borgnia, Zeyad Emam, Furong Huang, Micah Goldblum, and Tom Goldstein. End-to-end algorithm synthesis with recurrent networks: Extrapolation without overthinking. *Advances in Neural Information Processing Systems*, 35:20232–20242, 2022.

[6] Aviv Bick, Kevin Y Li, Eric P Xing, J Zico Kolter, and Albert Gu. Transformers to ssms: Distilling quadratic knowledge to subquadratic models. *Advances in neural information processing systems*, 37:31788–31812, 2024.

[7] Loïc Cabannes, Maximilian Beck, Gergely Szilvasy, Matthijs Douze, Maria Lomeli, Jade Copet, Pierre-Emmanuel Mazaré, Gabriel Synnaeve, and Hervé Jégou. Short window attention enables long-term memorization. *arXiv preprint arXiv:2509.24552*, 2025.

[8] Lucas Caccia, Alan Ansell, Edoardo Ponti, Ivan Vulić, and Alessandro Sordoni. Training plug-n-play knowledge modules with deep context distillation, 2025. URL https://arxiv.org/abs/2503.08727.

[9] Bowen Cao, Deng Cai, and Wai Lam. Infiniteicl: Breaking the limit of context window size via long short-term memory transformation, 2025. URL https://arxiv.org/abs/2504.01707.

[10] Mathieu Chalvidal, Thomas Serre, and Rufin VanRullen. Meta-reinforcement learning with self-modifying networks. *Advances in Neural Information Processing Systems*, 35:7838–7851, 2022.

[11] Tong Chen, Hao Fang, Patrick Xia, Xiaodong Liu, Benjamin Van Durme, Luke Zettlemoyer, Jianfeng Gao, and Hao Cheng. Generative adapter: Contextualizing language models in parameters with a single forward pass, 2024. URL https://arxiv.org/abs/2411.05877.

[12] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[13] Matthew Cook et al. Universality in elementary cellular automata. *Complex systems*, 15(1): 1–40, 2004.

[14] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, volume 2024, pages 35549–35562, 2024.

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

[16] Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

[17] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

[18] Xin Dong, Yonggan Fu, Shizhe Diao, Wonmin Byeon, Zijia Chen, Ameya Sunil Mahabaleshwarkar, Shih-Yang Liu, Matthijs Van Keirsbilck, Min-Hung Chen, Yoshi Suhara, et al. Hymba: A hybrid-head architecture for small language models. *arXiv preprint arXiv:2411.13676*, 2024.

[19] Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer. *arXiv preprint arXiv:1910.10073*, 2019.

[20] Sabri Eyuboglu, Ryan Ehrlich, Simran Arora, Neel Guha, Dylan Zinsley, Emily Liu, Will Tennien, Atri Rudra, James Zou, Azalia Mirhoseini, et al. Cartridges: Lightweight and general-purpose long context representations via self-study. *arXiv preprint arXiv:2506.06266*, 2025.

[21] Tao Ge, Jing Hu, Lei Wang, Xun Wang, Si-Qing Chen, and Furu Wei. In-context autoencoder for context compression in a large language model. *arXiv preprint arXiv:2307.06945*, 2023.

[22] Jonas Geiping, Sean Michael McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. In *NeurIPS*, 2025.

[23] Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[24] Yuxian Gu, Qinghao Hu, Shang Yang, Haocheng Xi, Junyu Chen, Song Han, and Han Cai. Jet-Nemotron: Efficient language model with post neural architecture search. *arXiv preprint arXiv:2508.15884*, 2025.

[25] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory*. Psychology press, 2005.

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

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

[28] Anmol Kabra, Yilun Yin, Albert Gong, Kamile Stankeviciute, Dongyoung Go, Johann Lee, Katie Z. Luo, Carla P. Gomes, and Kilian Q. Weinberger. Learning from synthetic data improves multi-hop reasoning. In *International Conference on Learning Representations*, 2026.

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

[30] Kevin Lin, Charlie Snell, Yu Wang, Charles Packer, Sarah Wooders, Ion Stoica, and Joseph E Gonzalez. Sleep-time compute: Beyond inference scaling at test-time. *arXiv preprint arXiv:2504.13171*, 2025.

[31] Bingbin Liu, Jordan T Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. Transformers learn shortcuts to automata. *arXiv preprint arXiv:2210.10749*, 2022.

[32] Yuxi Liu, Konpat Preechakul, Kananart Kuwaranancharoen, and Yutong Bai. The serial scaling hypothesis. *arXiv preprint arXiv:2507.12549*, 2025.

[33] James L McClelland, Bruce L McNaughton, and Randall C O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

[34] Sean McLeish, Ang Li, John Kirchenbauer, Dayal Singh Kalra, Brian R Bartoldson, Bhavya Kailkhura, Avi Schwarzschild, Jonas Geiping, Tom Goldstein, and Micah Goldblum. Teaching pretrained language models to think deeper with retrofitted recurrence. *arXiv preprint arXiv:2511.07384*, 2025.

[35] William Merrill and Ashish Sabharwal. A little depth goes a long way: The expressive power of log-depth transformers. *arXiv preprint arXiv:2503.03961*, 2025.

[36] Ida Momennejad, A Ross Otto, Nathaniel D Daw, and Kenneth A Norman. Offline replay supports planning in human reinforcement learning. *elife*, 7:e32548, 2018.

[37] Turlough Neary and Damien Woods. P-completeness of cellular automaton rule 110. In *International Colloquium on Automata, Languages, and Programming*, pages 132–143. Springer, 2006.

[38] Shahriar Noroozizadeh, Vaishnavh Nagarajan, Elan Rosenfeld, and Sanjiv Kumar. Deep sequence models tend to memorize geometrically; it is unclear why. *arXiv preprint arXiv:2510.26745*, 2025.

[39] NVIDIA. NVIDIA Nemotron Nano 2: An accurate and efficient hybrid Mamba-Transformer reasoning model. *arXiv preprint arXiv:2508.14444*, 2025.

[40] Hayden Prairie, Zachary Novack, Taylor Berg-Kirkpatrick, and Daniel Y Fu. Parcae: Scaling laws for stable looped language models. *arXiv preprint arXiv:2604.12946*, 2026.

[41] Björn Rasch and Jan Born. About sleep's role in memory. *Physiological reviews*, 2013.

[42] Liliang Ren, Yang Liu, Yadong Lu, Yelong Shen, Chen Liang, and Weizhu Chen. Samba: Simple hybrid state space models for efficient unlimited context language modeling. *arXiv preprint arXiv:2406.07522*, 2024.

[43] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International conference on machine learning*, pages 9355–9366. PMLR, 2021.

[44] Avi Schwarzschild, Eitan Borgnia, Arjun Gupta, Furong Huang, Uzi Vishkin, Micah Goldblum, and Tom Goldstein. Can you learn an algorithm? generalizing from easy to hard problems with recurrent networks. *Advances in Neural Information Processing Systems*, 34:6695–6706, 2021.

[45] Kristian Schwethelm, Daniel Rueckert, and Georgios Kaissis. How much is one recurrence worth? iso-depth scaling laws for looped language models. *arXiv preprint arXiv:2604.21106*, 2026.

[46] Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context, 2022. URL https://arxiv.org/abs/2209.15189.

[47] Jihoon Tack, Jaehyung Kim, Eric Mitchell, Jinwoo Shin, Yee Whye Teh, and Jonathan Richard Schwarz. Online adaptation of language models with a memory of amortized contexts, 2024. URL https://arxiv.org/abs/2403.04317.

[48] Arnuv Tandon, Karan Dalal, Xinhao Li, Daniel Koceja, Marcel Rød, Sam Buchanan, Xiaolong Wang, Jure Leskovec, Sanmi Koyejo, Tatsunori Hashimoto, et al. End-to-end test-time training for long context. *arXiv preprint arXiv:2512.23675*, 2025.

[49] Qwen Team. Qwen3.5: Accelerating productivity with native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

[50] Edward C Tolman. Cognitive maps in rats and men. *Psychological review*, 55(4):189, 1948.

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

[52] Junxiong Wang, Daniele Paliotta, Avner May, Alexander M Rush, and Tri Dao. The mamba in the llama: Distilling and accelerating hybrid models. *Advances in Neural Information Processing Systems*, 37:62432–62457, 2024.

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

[54] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024.

[55] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *Advances in neural information processing systems*, 37:115491–115522, 2024.

[56] Zeliang Zhang, Xiaodong Liu, Hao Cheng, Hao Sun, Chenliang Xu, and Jianfeng Gao. Training large reasoning models efficiently via progressive thought encoding. *arXiv preprint arXiv:2602.16839*, 2026.

[57] Yang Zhou, Hongyi Liu, Zhuoming Chen, Yuandong Tian, and Beidi Chen. GSM-Infinite: How do your LLMs behave over infinitely increasing reasoning complexity and context length? In *ICML 2025 Workshop on Long-Context Foundation Models*, 2025.

[58] Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, He Xing, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.