LATTICE: LEARNING TO EFFICIENTLY COMPRESS THE MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Attention mechanisms have revolutionized sequence learning but remain bottlenecked by quadratic computational complexity. This paper introduces Lattice, a novel recurrent neural network (RNN) mechanism that leverages the inherent low-rank structure of key-value matrices to efficiently compress contextual information into a fixed number of memory slots, achieving sub-quadratic complexity. By framing this compression as an online optimization problem, we derive a dynamic memory update rule based on a single gradient descent step. The resulting recurrence features a state- and input-dependent gating mechanism, offering an interpretable and theoretically grounded memory update process. The core innovation is the orthogonal update: each memory slot is updated exclusively with information orthogonal to its current state, hence incorporating only novel, non-redundant data to minimize interference with previously stored information. To ensure hardware-efficient training, we derive specialized chunk-wise parallelization forms tailored to this non-linear recurrence. Empirically, Lattice outperforms strong baselines on language modeling and associative recall tasks across diverse context lengths and model sizes, achieving superior memory efficiency.

1 INTRODUCTION

The attention mechanism (Vaswani et al., 2017b) has become a cornerstone of sequence modeling, seeing widespread adoption across domains, including natural language processing (NLP) (Devlin et al., 2018; Radford et al., 2018), computer vision (Dosovitskiy et al., 2020; Arnab et al., 2021), and graph structure learning and generation (Yun et al., 2019; Dwivedi & Bresson, 2020; Karami, 2024). However, despite its effectiveness, the quadratic time and space complexity of attention limits its scalability in long sequence modeling. Additionally, its reliance on an unbounded cache leads to inefficient memory management, further limiting its applicability in resource-constrained settings. These challenges have motivated the development of alternative architectures that aim to retain the expressivity of Transformers while addressing their computational bottlenecks.

Sequence mixing approaches like state space models (SSMs) (Gu et al., 2021; 2020; 2022; 2020; Mehta et al., 2022) and linear attention variants (Katharopoulos et al., 2020; Choromanski et al., 2020) have recently gained renewed interest as promising alternatives to softmax attention. While traditional SSMs, with their inherent linear recurrent structure, offer parallelization during training, they often struggle to match the expressivity of standard attention. More recently, input-dependent SSMs (Gu & Dao, 2023; Dao & Gu, 2024) and modern gated linear RNNs (Orvieto et al., 2023; De et al., 2024; Beck et al., 2024; Peng et al., 2025; Yang et al., 2024a) have demonstrated enhanced expressiveness and improved in-context learning while enabling parallelization through techniques like the associative scan (Blelloch, 1990; Smith et al., 2023; De et al., 2024). However, a fundamental challenge remains: while their memory is inherently constrained by a fixed-size hidden state (Arora et al., 2024), their linear updates either lack selective interaction or rely on purely linear interactions—such as the delta rule (Widrow & Hoff, 1988)—between stored information and incoming keys. Consequently, this limits their ability to perform complex filtering and discard redundant content dynamically. To overcome these representational bottlenecks, recent works have revisited non-linear recurrent networks (Beck et al., 2024; Sun et al., 2024) to build more expressive sequence models. Alternatively, global convolutions (Romero et al., 2021; Li et al., 2022; Poli et al., 2023) and their input-dependent variants (Karami et al., 2019; Karami & Ghodsi, 2024) offer another direction for long context

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

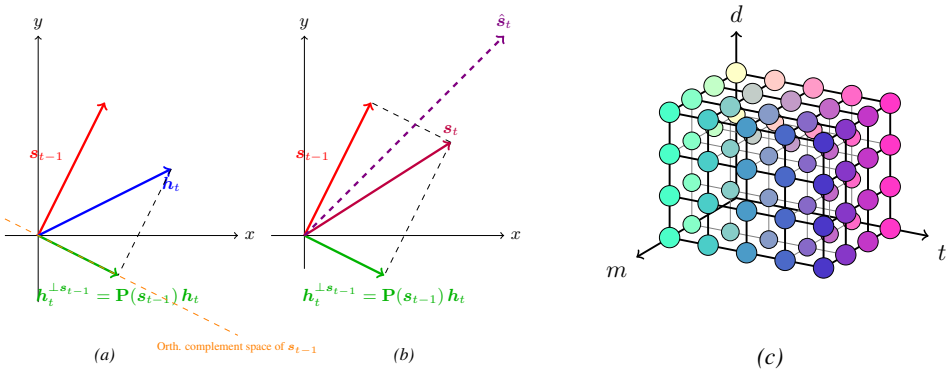

Figure 1: A geometric visualization of the proposed update rule. *(a)* A single current state vector, $\boldsymbol{s}_{t-1} = \mathbf{S}_{t-1}[:, i]$, an incoming token representation, $\boldsymbol{h}_t$, and its component orthogonal to the current state, $\boldsymbol{h}_t^{\perp \boldsymbol{s}_{t-1}}$. *(b)* Comparison of the updated state according to the proposed update rule ($\boldsymbol{s}_t = \boldsymbol{s}_{t-1} + \alpha_{i,t}\, \boldsymbol{h}_t^{\perp \boldsymbol{s}_{t-1}}$) and the updated state resulting from the superposition recurrence update of the standard linear attention ($\hat{\boldsymbol{s}}_t = \boldsymbol{s}_{t-1} + \alpha_{i,t}\, \boldsymbol{h}_t$, shown with a dashed arrow). For simplicity, a unit writing intensity ($\alpha_{i,t} = 1$) is assumed in both recurrent update rules. *(c)* Visualization of the relationships between $d \times m$ state matrices over time in state-dependent compression, depicted as interconnections of nodes in a 3D lattice. Each memory slot (state vector) is represented by a unique color.

modeling by dynamically adapting convolutional filters to the input, but they are not inherently compatible with causal modeling used in autoregressive generation.

Summary of contributions. We propose **Lattice**, a novel approach designed to address quadratic complexity of the attention layers. Our method compresses the cache into a fixed number of slots by leveraging the inherent low-rank structure of K-V matrices within an online optimization framework. This allows us to derive efficient, recursive memory update rules conditioned on its existing state and the current token with sub-quadratic complexity. In contrast to existing linear RNNs, which often rely on heuristics for memory management and lack explicit optimization for compression, we formulate memory compression as an optimization problem inspired by dictionary learning, from which we derive an interpretable and expressive non-linear recurrence. Lattice updates each memory slot exclusively with non-redundant information, specifically by incorporating only the component of the input that is orthogonal to the current state of that memory slot. To ensure scalability, we derive efficient, chunk-wise parallel forms of this specialized update rule. Finally, we empirically demonstrate the superior memory efficiency and language modeling capabilities of Lattice.

2 Background

Given an input sequence $\mathcal{X}_T = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T]$ with $\boldsymbol{x}_t \in \mathbb{R}^d$, the causal softmax attention mechanism generates output tokens $\boldsymbol{y}_t \in \mathbb{R}^d$, by attending to past tokens as:

$$\boldsymbol{y}_t = \mathcal{V}_t \, \texttt{Softmax}(\mathcal{K}_t^\top \, \boldsymbol{q}_t) \, . \tag{1}$$

Here, the queries, keys, and values are computed by linear projections: $\boldsymbol{q}_t = \mathbf{W}_q \, \boldsymbol{x}_t$, $\boldsymbol{k}_t = \mathbf{W}_k \, \boldsymbol{x}_t$, $\boldsymbol{v}_t = \mathbf{W}_v \, \boldsymbol{x}_t$, where $\mathbf{W}_q, \mathbf{W}_k \in \mathbb{R}^{m \times d}$ and $\mathbf{W}_v \in \mathbb{R}^{d \times d}$ are learnable weight matrices. The key-value cache, represented by $\mathcal{K}_t \in \mathbb{R}^{m \times t}$ and $\mathcal{V}_t \in \mathbb{R}^{d \times t}$, stacks the key and value vectors of each new token, resulting in a memory footprint that scales linearly with time. The retrieval of relevant information from this key-value cache can be rewritten as a weighted sum: $\boldsymbol{y}_t = \mathcal{V}_t \, \boldsymbol{a}_t$, where $\boldsymbol{a}_t = \texttt{Softmax}(\mathcal{K}_t^\top \, \boldsymbol{q}_t) \in \mathbb{R}^t$ represents the attention scores, capturing the correlations between the current token at step t and its historical context (past tokens). Consequently, the attention mechanism in equation 1 can be interpreted as performing a non-linear query over an unbounded memory. The linear growth of the key-value cache creates a significant memory bottleneck during inference, particularly for long sequences. Moreover, each retrieval operation scales linearly with sequence length, resulting in an overall quadratic computational complexity $\mathcal{O}(T^2)$ for generating a sequence of length T.

To address the computational and memory bottlenecks of the Softmax attention, various alternatives have been proposed (Tay et al., 2022). A well-established approach involves employing the kernel trick

to replace the softmax operation with a dot product of feature maps, $\phi(\boldsymbol{q}_t)$ and $\phi(\boldsymbol{k}_t)$, (Katharopoulos et al., 2020), commonly known as *linear attention* (LA). This mechanism can be formulated as: $\boldsymbol{y}_t = \left(\sum_{i=1}^{t} \boldsymbol{v}_i \phi(\boldsymbol{k}_i)^\top\right) \phi(\boldsymbol{q}_t)$, which can be expressed as the following linear recurrent model, also known as an input-dependent state-space model (SSM)[1]:

$$\{\boldsymbol{y}_t\}_{t=1}^{T} = \mathrm{LA}(\{\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t\}_{t=1}^{T}) := \begin{cases} \mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t\, \boldsymbol{k}_t^\top, & \textit{recurrence} \\ \boldsymbol{y}_t = \mathbf{S}_t \boldsymbol{q}_t & \textit{memory read-out} \end{cases}$$

This representation employs a simple linear recurrence to update the matrix-valued state $\mathbf{S}_t$, which compactly encodes key-value associations memory at each time step. Importantly, the linearity is key to achieving sub-quadratic parallel computation during training, using methods such as chunkwise computation (Hua et al., 2022; Kacham et al., 2024) or parallel scan (Blelloch, 1990; Smith et al., 2023), while retaining a per-token constant-time complexity at inference.

An alternative strategy to ensure bounded computational and memory requirements is to maintain a fixed-size key-value cache, where the memory matrices $\mathbf{K} \in \mathbb{R}^{m \times \tau}, \mathbf{V} \in \mathbb{R}^{d \times \tau}$ are constrained to a fixed length $\tau \ll T$. Sliding window attention which retains the most recent m tokens via a *first-in-first-out* (FIFO) queue is a simple form of this category. While computationally efficient, sliding window attention suffers from a limited receptive field which restricts the model's ability to capture long-range dependencies and maintain global context, resulting in a poor recall-memory trade-off (Arora et al., 2024). On the other hand, a growing body of research has observed that the key-value matrices in the attention layers often exhibit structured low-rank properties (Wang et al., 2020; Chen et al., 2021; Singhania et al., 2024). This insight suggests that instead of naively truncating memory, we can develop *efficient compression* techniques that selectively store the essential context while discarding less relevant or redundant information.

The update rule in the linear attention, and its gated variant, typically relies on an additive outer product of input-dependent representations, which can be expressed as: $\mathbf{S}_t = \mathbf{S}_{t-1} + f_v(\boldsymbol{x}_t) \otimes f_g(\boldsymbol{x}_t)$ where $f_v(\boldsymbol{x}_t)$ is an embedding of the input token and f_g can be interpreted as an input gate that controls the writing intensity.[2] While this *linear rank-one modification* to the state matrix (*a.k.a.* Hebbian-like update rule (Hebb, 2005)) enables efficient parallel computation, it suffers from a key limitation: the additive update term in the recurrence is agnostic to the current state $\mathbf{S}_{t-1}$ and operates independently of it. This lack of state awareness can cause *key interference* and eventually lead to an *overcapacity regime* (Schlag et al., 2021), where multiple tokens attempt to write to the same memory slot when the memory size is smaller than the sequence length.

Based on this insight, ideally, the writing intensity of the t-th token $\boldsymbol{x}_t$ to the i-th memory slot, $\boldsymbol{s}_i = \mathbf{S}[:, i]$, should depend on the interaction between the new token itself and the content of that slot. From a gating perspective, the gating mechanism should have access to the current state of the memory to make informed decisions about which information to add or discard (Hochreiter et al., 1997; Gers et al., 2002). This requires a *state-dependent gating* mechanism that dynamically modulates updates based on the current memory state. Although naive state conditioning can break the ability to parallelize computations, we address this computational bottleneck by adapting chunk-wise parallelization techniques, deriving approximations tailored specifically to our proposed recurrence, as detailed later in the manuscript. In the following section, we frame the role of the recurrent layer as solving an online optimization problem and derive an optimal update rule to compress and retain essential information from a sequence.

3 Compression Layer

State-Dependent Compression for Unbounded Caches. Our goal is to develop a compression model that dynamically updates and maintains a compact representation of the contextual history—conventionally stored in the key and value caches of a transformer model—in a streaming manner. As new tokens arrive, the model selectively distills and stores essential contextual information into a compressed memory matrix. This enables efficient querying, as the memory read-out is

[1] Consistent with many linear attention models, we omit the normalization term here to avoid potential numerical instabilities (Qin et al., 2022). Furthermore, we employ an identity mapping as the feature map, effectively absorbing any transformation into the corresponding projection layers.

[2] These two are also called *role* and *filler* vectors in tensor product representation (Smolensky, 1990).

processed using the compressed state, i.e. $\boldsymbol{y}_t = \mathbf{S}_t \hat{\boldsymbol{r}}_t$, instead of querying the full cache. Here, $\hat{\boldsymbol{r}}_t$ represents a retrieval vector analogous to the attention weights in standard attentions layers.

This lossy compression approach entails a trade-off between computational efficiency, memory usage, and query precision. A more compact memory representation (i.e., smaller m) reduces both computational cost and memory footprint, but at the expense of information loss and lower fidelity in reconstructing the original context, thereby diminishing the overall expressivity of the model. We aim to design an optimal lossy compression layer that minimizes this precision loss. We formulate this problem as an input reconstruction task, where we enforce: $\mathbf{x}_t \approx \tilde{\mathbf{x}}_t = \mathbf{S}_t \boldsymbol{k}_t$. Here, $\tilde{\mathbf{x}}_t$ is the reconstructed input, $\mathbf{S}_t$ represents the dynamically updated state, and $\boldsymbol{k}_t$ is a latent representation vector[3].

Inspired by classical representation learning techniques such as dictionary learning, sparse coding, and structured matrix factorization (Mairal et al., 2009; Lyu et al., 2020)[4], we interpret our approach as dynamically learning and updating basis vectors (a.k.a. dictionary atoms) and their corresponding latent coefficients (analogous to sparse codes).

3.1 Decoding Layer

For each input sequence, we model a *decoding layer*, denoted as $g(\boldsymbol{k}_t; \mathbf{S}_t)$, which operates on the latent representation $\boldsymbol{k}_t$ and is parameterized by the state matrix $\mathbf{S}_t$. Unlike standard neural network layers, here we aim to dynamically update $\mathbf{S}_t$, over the course of a sequence, thereby effectively memorizing and encoding the historic context up to time t. Functionally, this acts as a decoding layer with an *internal state*, or equivalently, a *fast decoding layer*. Specifically, each token embedding $\boldsymbol{v}_t$ is paired with its corresponding latent representation (code) $\boldsymbol{k}_t$, and the decoding function $g(\boldsymbol{k}_t; \mathbf{S}_t)$ aims to reconstruct $\boldsymbol{v}_t$. To achieve this, we formulate an optimization problem that minimizes a loss function $\ell(\cdot, \cdot)$—quantifying the dissimilarity between the decoded output and the target vector—as its objective at each time step:

$$\mathcal{L}_t = \ell\big(g(\boldsymbol{k}_t; \mathbf{S}_t),\ \boldsymbol{v}_t\big), \tag{2}$$

where $\mathbf{S}_t \in \mathbb{R}^{d \times m}$, $\boldsymbol{v}_t \in \mathbb{R}^d$, $\boldsymbol{k}_t \in \mathbb{R}^m$. We refer to this objective as the *compression loss* throughout this paper. Here, the latent representation $\boldsymbol{k}_t$ is generated by a model-based encoder network, modeled simply as a linear projection of the input: $\boldsymbol{k}_t = \mathbf{W}_k \mathbf{x}_t$ where $\mathbf{W}_k \in \mathbb{R}^{m \times d}$ is a projection weight matrix. This weight remains fixed during the internal state updates and is trained jointly with the rest of the model parameters in the outer training loop. This setup follows the well-established meta-learning frameworks (Schmidhuber, 1992; Thrun & Pratt, 1998; Andrychowicz et al., 2016; Sun et al., 2024) or bilevel optimization approaches (Liu et al., 2022; Chen et al., 2022).

The proposed framework consists of two distinct types of parameters: (I) the internal states of the compression layers, $\mathbf{S}_t$, which dynamically store in-context information for each sequence, and (II) the outer model parameters, including the projection layer weights, collectively denoted as $\boldsymbol{\mathcal{W}}$, which capture broader patterns across the training set. This leads to a bilevel learning process composed of:

- *Inner Loop (State Update):* A fast update mechanism that adapts the internal states $\mathbf{S}_t$ for each token within a sequence by minimizing the compression loss in equation 2. Each sequence effectively serves as a distinct dataset for the inner loop, which encodes in-context information into a sequence of evolving states $\{\mathbf{S}_t\}_{t=1}^T$. Throughout this process, the outer model weights, $\boldsymbol{\mathcal{W}}$, remain frozen.
- *Outer Training Loop:* The standard training of the neural network that learns $\boldsymbol{\mathcal{W}}$ by minimizing the average loss across all training sequences for the (self-)supervised learning task. This slower loop typically employs standard optimizers such as ADAM (Kingma & Ba, 2014) to learn generalizable patterns from the training dataset.

[3]Note that while the objective reconstructs from $\mathbf{S}_t$ only the current time step $\boldsymbol{x}_t$, due to the online gradient descent form of how this objective is minimized, the state $\mathbf{S}_t$ implicitly compresses the history, allowing it to reconstruct the entire sequence, i.e. $\mathbf{S}_t \boldsymbol{k}_k \approx \boldsymbol{x}_k$. Since $\mathbf{S}_t$ has the same interpretation as the state in RNNs and SMMs, the same notation is reused.

[4]This problem has been studied under various names over the decades, including dictionary learning, factor analysis, topic modeling, and component analysis, each with slightly different constraints and emphases (Lyu et al., 2020).

From an optimization perspective, this bilevel process is analogous to alternating optimization (Goldstein et al., 2014), where the inner loop optimizes the state $\mathbf{S}_t$ while keeping $\boldsymbol{\mathcal{W}}$ fixed, and the outer loop optimizes $\boldsymbol{\mathcal{W}}$ based on the adapted states.

This work focuses on designing an optimal update rule for the memory states. Given the streaming nature of sequence data, a standard approach is to treat the inner loop as an online regression problem and employ the steepest descent method. Specifically, the internal state is dynamically updated using a single gradient descent step per token:

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \gamma_t \nabla_S \mathcal{L}(\mathbf{S}_{t-1}, \boldsymbol{v}_t, \boldsymbol{k}_t) \tag{3}$$

This recursive update yields a sequence of states $\{\mathbf{S}_t\}_{t=1}^T$, where each new state $\mathbf{S}_t$ is a nonlinear function of the current state and the current input token, ensuring a causal and context-dependent evolution of the internal state.

3.1.1 State Normalization

Following established practices in dictionary and subspace learning—where basis vectors (a.k.a. dictionary atoms or principal components) are normalized—we apply column-wise normalization to each vector of the state matrix. The decoding function $g(\boldsymbol{k}_t; \mathbf{S}_t)$ maps the latent representation $\boldsymbol{k}_t \in \mathbb{R}^m$ back to the reconstructed target $\boldsymbol{v}_t \in \mathbb{R}^d$ using the normalized state matrix $\phi(\mathbf{S}_t)$: $\hat{\boldsymbol{v}}_t = g(\boldsymbol{k}_t; \mathbf{S}_t) = \phi(\mathbf{S}_t)\boldsymbol{k}_t$. At each time step, the internal states are updated to minimize the discrepancy between the linear combination of these normalized basis vectors and the target vector $\boldsymbol{v}_t$. We consider two primary objectives for this compression.

First, we adopt the standard ℓ_2 reconstruction loss, measuring the squared Euclidean distance between the decoded output and the target:

$$\mathcal{L}_t = \frac{1}{2}\|\phi(\mathbf{S}_t)\boldsymbol{k}_t - \boldsymbol{v}_t\|^2, \tag{4}$$

where $\mathbf{S}_t \in \mathbb{R}^{d\times m}$, $\boldsymbol{v}_t \in \mathbb{R}^d$, $\boldsymbol{k}_t \in \mathbb{R}^m$. To derive the closed-form gradient of this objective, let $\boldsymbol{s}_i$ denote the i-th column (i-th basis vector) of $\mathbf{S}_{t-1}$, and let $\boldsymbol{\phi}_i = \frac{\boldsymbol{s}_i}{\|\boldsymbol{s}_i\|}$ be its normalized counterpart. The reconstruction error is defined as $\boldsymbol{e}_t := \phi(\mathbf{S}_{t-1})\boldsymbol{k}_t - \boldsymbol{v}_t$. The Jacobian of the normalization function $\phi(\boldsymbol{s}_i) = \frac{\boldsymbol{s}_i}{\|\boldsymbol{s}_i\|}$ is given by:

$$\mathbf{J}_\phi(\boldsymbol{s}_i) = \frac{1}{\|\boldsymbol{s}_i\|}\left(\mathbf{I} - \frac{\boldsymbol{s}_i\boldsymbol{s}_i^\top}{\|\boldsymbol{s}_i\|^2}\right) = \frac{\mathbf{P}(\boldsymbol{s}_i)}{\|\boldsymbol{s}_i\|}.$$

Applying the chain rule, the gradient of the loss with respect to each memory slot $\boldsymbol{s}_i$ is:

$$\nabla_{\boldsymbol{s}_i}\mathcal{L}_t = k_{i,t}\frac{\mathbf{P}(\boldsymbol{s}_i)}{\|\boldsymbol{s}_i\|}\boldsymbol{e}_t \tag{5}$$

Alternatively, the compression objective can be formulated to maximize the dot-product similarity between the decoded output and target representation:

$$\mathcal{L}_t = -\langle \phi(\mathbf{S}_t)\boldsymbol{k}_t,\ \boldsymbol{v}_t\rangle, \tag{6}$$

The gradient for this objective is derived similarly:

$$\nabla_{\boldsymbol{s}_i}\mathcal{L}_t = -k_{i,t}\frac{\mathbf{P}(\boldsymbol{s}_i)}{\|\boldsymbol{s}_i\|}\boldsymbol{v}_t \tag{7}$$

This gradient derivation reveals a *highly interpretable and interesting update rule*. The matrix $\mathbf{P}(\boldsymbol{s}_i) = \mathbf{P}(\boldsymbol{\phi}_i) := \left(\mathbf{I} - \frac{\boldsymbol{s}_i\boldsymbol{s}_i^\top}{\|\boldsymbol{s}_i\|^2}\right)$, which appears in (5) and (7), is formally known as the *projection matrix onto the orthogonal complement* of $\boldsymbol{s}_i$ in linear algebra (Strang, 2000, §3.3). This insight implies that the update for each memory slot (column $\boldsymbol{s}_i$) is driven by a projection of the input vector (e.g., $\boldsymbol{h}_t = -\boldsymbol{v}_t$ in (7) or $\boldsymbol{h}_t = \boldsymbol{e}_t$ in (5)) onto the space orthogonal to that slot. This suggests an interpretable decomposition of the $\boldsymbol{h}_t$ into two components: I) $\boldsymbol{h}_t^{\perp \boldsymbol{s}_i}$, *the component orthogonal to* $\boldsymbol{s}_i$, which is *used* to update the memory slot. II) $\boldsymbol{h}_t^{\parallel \boldsymbol{s}_i}$, the component of $\boldsymbol{h}_t$ aligned with $\boldsymbol{s}_i$, which is *discarded* in the update rule, ensuring non-redundant updates. This implies that *each memory slot is updated only with new information that is not already captured in that slot*. The scalar $k_{i,t}$ acts as a *writing intensity*, determining the t-th token's contribution to the i-th memory slot. This orthogonal update process is visualized in Figure 1.

Therefore, applying online gradient descent (OGD) (3) to the compression losses offers a principled approach for deriving a recurrent update rule based on the orthogonal projection onto the current state. By defining $\boldsymbol{h}_t := \boldsymbol{e}_t$ for the ℓ_2 loss or $\boldsymbol{h}_t := -\boldsymbol{v}_t$ for the dot-product similarity objective, these updates can be unified into a general rule, the *Orthogonal State Recurrence (OSR)* for the state matrix:

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \gamma_t \Big(\Big[\frac{\mathbf{P}(\boldsymbol{s}_1)}{\|\boldsymbol{s}_1\|}, \ldots, \frac{\mathbf{P}(\boldsymbol{s}_m)}{\|\boldsymbol{s}_m\|} \Big] \times_2 \boldsymbol{h}_t \Big) \odot \boldsymbol{k}_t^\top \tag{8}$$

where $\odot$ denotes the element-wise (Hadamard) product with broadcasting, and $\times_2$ is a vector-tensor product defined as $\mathcal{P}(\mathbf{S}) \times_2 \boldsymbol{h} := [\mathcal{P}_{:,:,1}\boldsymbol{h}, \ldots, \mathcal{P}_{:,:,m}\boldsymbol{h}]$ where the tensor $\mathcal{P}(\mathbf{S}) := \Big[\frac{\mathbf{P}(\boldsymbol{s}_1)}{\|\boldsymbol{s}_1\|}, \ldots, \frac{\mathbf{P}(\boldsymbol{s}_m)}{\|\boldsymbol{s}_m\|} \Big] \in \mathbb{R}^{d \times d \times m}$ is formed by stacking the normalized projectors along the last dimension[5]. A simplified hardware-efficient form of the update is presented in §3.4 and an alternative *encoding representation* is detailed in Appendix C.2. Moreover, we compare the proposed OSR updates of the compression layers with the OGD-based recurrences of existing RNNs in Table 3.

3.2 Stabilizing Memory Updates via Normalization

At each recurrence step, a memory slot is updated by incorporating only the component of the new information that is *orthogonal* to its current state. Formally, we update the i-th memory slot as $\boldsymbol{s}_{i,t} = \boldsymbol{s}_{i,t-1} + \Delta\boldsymbol{s}_{i,t}$, where $\Delta\boldsymbol{s}_{i,t} := c_{i,t}\, \boldsymbol{h}_t^{\perp \boldsymbol{s}_{i,t-1}}$, with $\boldsymbol{h}_t^{\perp \boldsymbol{s}_{i,t-1}}$ denoting the orthogonal component of the input relative to $\boldsymbol{s}_{i,t-1}$ and $c_{i,t}$ representing an input-dependent writing intensity. While this update scheme avoids interfering with the existing memory by *adding only novel, non-redundant information*, it inherently leads to a monotonic increase in the norm of $\boldsymbol{s}_i$ with each update, as dictated by the Pythagorean theorem: $\|\boldsymbol{s}_{i,t}\|^2 = \|\boldsymbol{s}_{i,t-1}\|^2 + \|\Delta\boldsymbol{s}_{i,t}\|^2$. This unbounded growth can induce numerical instability and state magnitude explosion or may dilute the effective representation of information over time.

To address this issue, we constrain the feasible set for the state vectors to the unit sphere $\mathcal{C} = \{\boldsymbol{s} \in \mathbb{R}^d \mid \|\boldsymbol{s}\| = 1\}$, and enforce this constraint by projecting the resulting Euclidean update back onto $\mathcal{C}$, denoted by $\mathcal{P}_{\mathcal{C}}(\cdot)$, at each time step. Therefore, the effective update becomes

$$\boldsymbol{s}_{i,t} = \mathcal{P}_{\mathcal{C}}(\boldsymbol{s}_{i,t-1} + \Delta\boldsymbol{s}_{i,t}) = \beta_{i,t}\,(\boldsymbol{s}_{i,t-1} + \Delta\boldsymbol{s}_{i,t})\,, \tag{9}$$

where $\beta_{i,t} = \big(1 + \|\Delta\boldsymbol{s}_{i,t}\|^2\big)^{-\frac{1}{2}}$, assuming $\|\boldsymbol{s}_{i,t-1}\| = 1$. This normalization, achieved by scaling with $\beta_{i,t}$, ensures that the updated state $\boldsymbol{s}_{i,t}$ remains on the unit sphere while preserving the steepest-descent direction, thereby maintaining stability and allowing the model to effectively store relevant information. Functionally, this normalization of the recurrence terms *inherently induces a state-dependent forget gate* in RNNs and adaptively *normalizes the effective step size of the update term*—a technique known to improve convergence in optimization algorithms like Adagrad (Duchi et al., 2011) and Adam (Kingma & Ba, 2014). We formalize the relationship between the proposed *Normalized Orthogonal State Recurrence* (NOSR) and Riemannian optimization (Absil et al., 2009; Boumal, 2023), as follows:

Proposition 3.1 (Equivalence to Riemannian GD) *Let $\mathcal{C} = \{\boldsymbol{s} \in \mathbb{R}^d \mid \|\boldsymbol{s}\| = 1\}$ denote the unit sphere. Then, the projected gradient update of the form $\boldsymbol{s}_{i,t} = \mathcal{P}_{\mathcal{C}}(\boldsymbol{s}_{i,t-1} + \Delta\boldsymbol{s}_{i,t})$ (as in equation 9), where the update term $\Delta\boldsymbol{s}_{i,t}$ lies in the subspace orthogonal to $\boldsymbol{s}_{i,t-1}$ (cf. (8)), is equivalent to a retraction step in Riemannian optimization (Bonnabel, 2013).*

3.3 Forgetting by State Regularization

Similar to how regularization is used in standard neural network training to control the memorization of the model, we can apply regularization to the states in the inner loop to manage memory retention. Specifically, applying ℓ_2 regularization to the state matrix $\mathbf{S}_t$ yields the regularized objective function: $\hat{\mathcal{L}}_t = \frac{1}{2}\|\phi(\mathbf{S}_t)\,\boldsymbol{k}_t - \boldsymbol{v}_t\|^2 + \frac{\lambda_t}{2}\|\mathbf{S}_t\|_F^2$, where λ_t is the regularization parameter and $\|\cdot\|_F$ denotes the Frobenius norm. Optimizing this differentiable objective using gradient descent results in a recurrence with state decay for the decoding compression layer (equation 4):

$$\mathbf{S}_t = \mu_t \mathbf{S}_{t-1} - \gamma_t \nabla_S \mathcal{L}(\mathbf{S}_{t-1}, \boldsymbol{v}_t, \boldsymbol{k}_t), \tag{10}$$

where the scalar $\mu_t = 1 - \gamma_t\lambda_t \in [0,\ 1]$ acts as a forget gate, controlling the proportion of the past memory that is retained in the update.

[5]This operation can be implemented using Einstein sum as: `einsum("d1 d m, d->d1 m", J,h)`.

General Form. By integrating all components, we arrive at the non-linear state transition for the i-th memory slot[6]: $\boldsymbol{s}_{i,t} = \mu_t\,\beta_{i,t}\,\boldsymbol{s}_{i,t-1} - \gamma_t\,\beta_{i,t}\left(\mathbf{P}(\boldsymbol{s}_{i,t-1})\boldsymbol{h}_t\right)k_{i,t}$. The complete update rule and read-out for the Lattice layer can be expressed in matrix form as:

$$\text{Lattice}(\boldsymbol{k}_t,\boldsymbol{v}_t,\boldsymbol{q}_t):\begin{cases}\mathbf{S}_t = (\mu_t\mathbf{1}\boldsymbol{\beta}_t^\top)\odot\mathbf{S}_{t-1} - \gamma_t\big(\left[\mathbf{P}(\boldsymbol{s}_{1,t-1}),\ldots,\mathbf{P}(\boldsymbol{s}_{m,t-1})\right]\times_2\boldsymbol{h}_t\big)\odot(\boldsymbol{k}_t\odot\boldsymbol{\beta}_t)^\top\\ \boldsymbol{y}_t = \mathbf{S}_t\boldsymbol{q}_t\end{cases} \tag{11}$$

where $\boldsymbol{\beta}_t \in \mathbb{R}^m$ is the per-slot normalization factor. We provide a detailed comparison of the proposed orthogonal state update with the standard linear delta-rule (Widrow & Hoff, 1988) and non-linear update rule of TTT (Sun et al., 2024) in Appendix B. In the following, we derive an efficient computation for Lattice including its chunk-wise parallelization.

3.4 Efficient Computation

The recurrence of equation 11 requires computing the projection $\boldsymbol{h}_t^{\perp\boldsymbol{s}_i} = \mathbf{P}\boldsymbol{h}_t$. Given the identity plus rank-one form of $\mathbf{P}$, the projection operation reduces to a vector dot product and a scalar-vector multiplication: $\boldsymbol{h}_t^{\perp\boldsymbol{s}_i} = \boldsymbol{h}_t - \boldsymbol{s}_i(\boldsymbol{s}_i^\top\boldsymbol{h}_t)$, avoiding a full matrix-vector multiplication. Hence, the computational cost of each projection scales linearly with the state dimension d, leading to an overall complexity of $\mathcal{O}(d\,m)$ for the full recurrence. By eliminating the need for vector-Jacobian products (`vjp`), we can express this explicit update in a hardware-efficient matrix form:

$$\mathbf{S}_t = \mathbf{G}_t\odot\mathbf{S}_{t-1} + \underbrace{\mathbf{1}(\hat{\boldsymbol{h}}_t\odot\hat{\boldsymbol{k}}_t)^\top\odot\mathbf{S}_{t-1} - \boldsymbol{h}_t\hat{\boldsymbol{k}}_t^\top}_{-\gamma_t\,(\mathbf{1}\boldsymbol{\beta}_t^\top)\odot\nabla_S\mathcal{L}(\mathbf{S}_{t-1},\boldsymbol{v}_t,\boldsymbol{k}_t)} \tag{12}$$

where $\mathbf{G}_t = \mu_t\mathbf{1}\boldsymbol{\beta}_t^\top \in \mathbb{R}^{d\times m}$, $\hat{\boldsymbol{h}}_t = \mathbf{S}_{t-1}^\top\boldsymbol{h}_t \in \mathbb{R}^m$ and $\hat{\boldsymbol{k}}_t = \gamma_t\boldsymbol{\beta}_t\odot\boldsymbol{k}_t \in \mathbb{R}^m$.

3.4.1 Parallel and Hardware Efficient Form

Various methods have been explored to enable the parallel evaluation of non-linear RNNs. One strategy casts inference as the solution to a fixed-point equation (Lim et al., 2023; Gonzalez et al., 2024). In a different approach, Sun et al. (2024) introduced a parallel chunk-wise solution using a gradient approximation. This method partitions a sequence into non-overlapping chunks of size C and utilizes the state at the beginning of each chunk (i.e., the final state from the preceding chunk)—denoted as $\mathbf{S}_{t'}$ where $t' = C\lfloor(t-1)/C\rfloor$ —to approximate the gradients for the entire chunk in parallel as: $\nabla_S\mathcal{L}(\mathbf{S}_{t'},\boldsymbol{v}_t,\boldsymbol{k}_t)$. While the approximate gradient of TTT reduces to a simple rank-one outer product ($\nabla_S\mathcal{L}_t \approx u_t k_t^\top$), leading directly to a standard chunk-wise parallelization similar to Linear Attention, the orthogonal projections in Lattice do not fit this standard form (cf. Appendix B for a detailed discussion). Consequently, we derive two specialized chunk-wise parallelization strategies for this recurrence.

By evaluating the entire gradient term at the chunk boundary $\mathbf{S}_{t'}$, we effectively apply a *zeroth-order* approximation of the gradient (w.r.t. $\mathbf{S}_{t-1}$) to linearize the recurrence in (12) within a chunk as:

$$\mathbf{S}_t = \mathbf{G}_t\odot\mathbf{S}_{t-1} + \mathbf{1}(\tilde{\boldsymbol{h}}_t\odot\hat{\boldsymbol{k}}_t)^\top\odot\mathbf{S}_{t'} - \boldsymbol{h}_t\hat{\boldsymbol{k}}_t^\top, \tag{13}$$

where $\tilde{\boldsymbol{h}}_t = \mathbf{S}_{t'}^\top\boldsymbol{h}_t$. To compute this efficiently over the time steps: $t = bC+\tau$ of the b-th chunk, let's stack the inputs into chunk-wise matrices, $\mathbf{X}^b = \boldsymbol{x}[bC+1:(b+1)C]$ (similarly for $\mathbf{Q}^b,\mathbf{H}^b,\mathbf{K}^b$). Defining the cumulative decay $\boldsymbol{a}_\tau^b = \prod_{i=bC+1}^{bC+\tau}\boldsymbol{\beta}_i\mu_i$, $\boldsymbol{A}^b = [\boldsymbol{a}_\tau^b]_{\tau=1}^C$, and a block lower-triangular tensor $\boldsymbol{\Omega}^b \in \mathbb{R}^{C\times C\times m}$ with components $\boldsymbol{\Omega}^b_{j,i,:} = \frac{\boldsymbol{a}_j^b}{\boldsymbol{a}_i^b}\mathbb{I}_{i\le j}$, the layer update at the chunk-level is expressed as:

$$\begin{aligned}\mathbf{S}_b &= \mathbf{1}(\boldsymbol{a}_C^b + \boldsymbol{f}^b)^\top\odot\mathbf{S}_{b-1} - \boldsymbol{H}^{b\top}(\hat{\boldsymbol{K}}^b\odot\boldsymbol{\Omega}^b_{C,:,:})\\ \boldsymbol{Y}^b &= \big(\boldsymbol{Q}^b\odot(\boldsymbol{A}^b+\boldsymbol{F}^b)\big)\mathbf{S}_{b-1}^\top - \boldsymbol{P}^b\boldsymbol{H}^b\end{aligned} \tag{14}$$

where $\boldsymbol{f}^b = \text{diag}[\tilde{\boldsymbol{H}}^{b\top}(\hat{\boldsymbol{K}}^b\odot\boldsymbol{\Omega}^b_{C,:})]$ and $\boldsymbol{F}^b_{in} = \sum_{\tau=1}^C(\tilde{\boldsymbol{H}}^b\odot\hat{\boldsymbol{K}}^b)_{\tau n}\,\boldsymbol{\Omega}^b_{i\tau n}$ (refer to Appendix C for the efficient computation of $\boldsymbol{f}^b$, $\boldsymbol{F}^b$ and $\boldsymbol{\beta}_\tau^b$). Here, $\boldsymbol{P}^b \in \mathbb{R}^{C\times C}$ is a lower-triangular matrix with elements $\boldsymbol{P}^b_{ij} = \sum_{k=1}^m\boldsymbol{Q}^b_{ik}\,\hat{\boldsymbol{K}}^b_{jk}\,\boldsymbol{\Omega}^b_{ijk}$. A `matmul`-optimal computation for $\boldsymbol{P}^b$ using a sub-tiling technique is presented by Zhang et al. (2024).

[6]For notational brevity, we hereafter assume $\|\boldsymbol{s}_{i,t-1}\|$=1, as maintained by the geometric normalization (9).

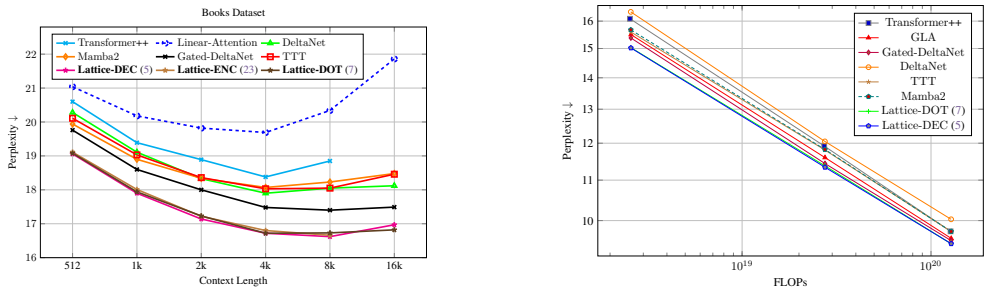

Figure 2: Scaling patterns. *(Left)*: Perplexity vs. context length (110M models, Books dataset). *(Right)*: Perplexity vs. model size (Chinchilla-style scaling, FineWeb-Edu). Transformer++ results are capped at $T \leq 8k$ since training them on very long contexts from scratch often performs poorly (Touvron et al., 2023).

A closer inspection of (12) reveals that we can retain greater expressivity by approximating only $\tilde{\boldsymbol{h}}_t = \mathbf{S}_{t'}^\top \boldsymbol{h}_t$ at the chunk boundary while preserving the *first-order* dependency of $\nabla_S \mathcal{L}$ on $\mathbf{S}_{t-1}$ as:

$$\mathbf{S}_t = \mu_t \mathbf{1}\boldsymbol{\beta}_t^\top \odot \mathbf{S}_{t-1} + \mathbf{1}(\tilde{\boldsymbol{h}}_t \odot \hat{\boldsymbol{k}}_t)^\top \odot \mathbf{S}_{t-1} - \boldsymbol{h}_t \hat{\boldsymbol{k}}_t^\top = \hat{\mathbf{G}}_t \odot \mathbf{S}_{t-1} - \boldsymbol{h}_t \hat{\boldsymbol{k}}_t^\top, \tag{15}$$

where $\hat{\mathbf{G}}_t = \mathbf{1}(\mu_t \boldsymbol{\beta}_t + \tilde{\boldsymbol{h}}_t \odot \hat{\boldsymbol{k}}_t)^\top$. Notably, by absorbing the state-dependent projection into $\hat{\mathbf{G}}_t$, the forget factor is structured as a rank-one matrix, which allows to efficiently compute the intra-chunk update by the parallel form presented in Gated Linear Attention (Zhang et al., 2024). As shown in Appendices C.1 and E, all operations in (14) and (15)—including their orthogonal projection and normalization—can be implemented using hardware-efficient tensor operations (`matmul` and $\odot$).

4 Experiments

Setup: We evaluate the proposed architecture across various language modeling tasks, benchmarking its performance on both short-context and long-context tasks. We trained models on different datasets: for language modeling and common-sense reasoning tasks, models are trained on the FineWeb-Edu dataset (Penedo et al., 2024) with a context length of 4k, for sequence length scaling analysis, models are trained on the Books3 dataset (a subset of The Pile dataset (Gao et al., 2020)) with different training context lengths ranging from 512 to 16k tokens (in increments of $2\times$ per experiment). Across all experiments, the training batch size is fixed at 0.5M tokens, irrespective of sequence length. We compare our method against the Transformer++ baseline (Touvron et al., 2023) as well as several state-of-the-art sub-quadratic sequence models: Linear-Attention (LA) (Katharopoulos et al., 2020), TTT (Sun et al., 2024), DeltaNet (Yang et al., 2024c), Gated DeltaNet (Yang et al., 2024a), Mamba2 (Dao & Gu, 2024) and GLA (Zhang et al., 2024). Detailed experimental settings, including a training throughput comparison (Figure 9), are provided in Appendix D.

The results in Figure 2 (and Figure 6) demonstrate that Lattice consistently achieves superior performance compared to all baselines across a range of context lengths. Importantly, the performance gains of Lattice relative to other linear RNNs—as detailed in Table 5—become increasingly pronounced as the sequence length grows. This scaling trend highlights the effectiveness of the proposed approach for long-context modeling where effective memory management is crucial. Furthermore, Figure 2 illustrates the scaling pattern with respect to increasing model size, showing that Lattice achieves lower perplexity than the baselines across different parameter scales.

Common-sense reasoning. We evaluate zero-shot accuracy on standard commonsense reasoning benchmarks—including PIQA (Bisk et al., 2020), HellaSwag (Hella.) (Zellers et al., 2019), WinoGrande (Wino.) (Sakaguchi et al., 2021), ARC-easy (ARC-e) and ARC-challenge (Arc-c) (Clark et al., 2018), and Commonsense QA (CSQA) (Talmor et al., 2018)— commonly used for LM benchmarking (Zhang et al., 2024; Yang et al., 2024c). As shown in Table 1, Lattice outperforms the baseline models on most of these reasoning tasks, achieving the highest average accuracy in both 340M and 760M parameter scales.

Furthermore, to assess memory capacity and in-context learning capabilities, we evaluate the model on the Multi-Query Associative Recall (MQAR) task (Arora et al., 2023). Results in Figure 3 and §D.2 demonstrate Lattice achieves significantly higher recall accuracy than compared recurrent baselines, with gains growing as sequence length increases. This highlights the efficiency of its compression mechanism in retaining dense information within a fixed-size recurrent memory.

Table 1: Performance comparison on zero-shot common-sense reasoning tasks. Models are trained on FineWeb-Edu dataset.

Model	PIQA acc ↑	Hella. acc_n ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	CSQA acc ↑	BoolQ acc ↑	Avg.
340M params / 15B tokens								
Transformer++	66.76	40.40	52.38	49.47	27.04	33.01	58.93	46.85
GLA	67.52	42.10	52.09	53.11	29.36	36.77	59.79	48.68
Mamba2	67.08	41.30	52.25	51.63	29.19	36.28	60.98	48.39
DeltaNet	66.21	40.80	**52.64**	50.74	27.73	36.20	**62.05**	48.05
TTT	66.59	41.10	51.86	52.14	26.67	35.71	60.89	47.88
Gated-DeltaNet	68.12	42.93	52.33	53.19	29.44	36.45	57.74	48.60
Lattice-DEC (5) C=1	**68.28**	**43.33**	51.70	53.66	28.41	37.27	60.70	49.05
Lattice-DEC (5) C=4	67.25	43.20	51.93	**54.33**	**29.70**	36.12	61.44	**49.14**
Lattice-DOT (7) C=4	67.95	43.13	51.07	53.91	28.50	**38.33**	57.71	48.66
760M params / 30B tokens								
Transformer++	69.59	51.13	53.83	56.96	30.30	39.64	**62.11**	51.94
GLA	69.64	52.17	53.20	59.07	32.62	41.20	60.12	52.57
Mamba2	70.18	50.67	52.64	59.28	33.99	41.77	57.19	52.25
DeltaNet	69.86	48.60	51.85	58.10	32.36	40.05	58.56	51.34
TTT	70.08	50.67	52.09	58.65	33.73	41.28	60.43	52.42
Gated-DeltaNet	71.38	51.87	53.99	**61.06**	34.08	39.97	56.18	52.64
Lattice-DEC (5) C=4	71.00	**52.93**	53.28	59.66	34.68	**43.90**	59.33	53.54
Lattice-DOT (7) C=4	**71.76**	51.70	**54.38**	59.49	**36.57**	43.57	60.73	**54.03**

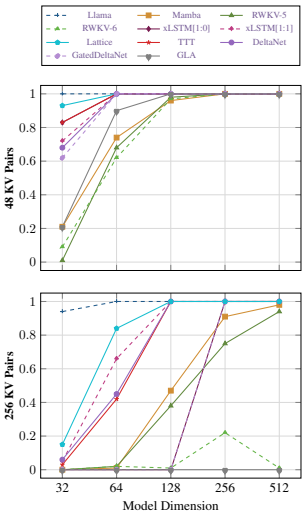

Figure 3: Associative Recall Accuracy (%) on the recall-intensive MQAR task. Performance is evaluated across different d for KV pair $\in \{48, 256\}$ and $T = 2048$.

Table 2: Ablation study of Lattice's components. Each row starting with + adds a new component to the configuration in the row above, beginning from the DeltaNet baseline. The final row with + represents the full Lattice configuration. The default memory size is set to $m = d = 64$. It also compares the two parallel approximations and different memory size m.

Configuration	ppl ↓
DeltaNet	16.35
TTT C=1	15.60
Lattice C=1	
+ orthogonal recurrence (5)	15.38
+ normalized projection (9)	15.07
+ forget-gate (10)	**15.02**
Lattice C=4, approx. (13)	15.15
Lattice C=4, approx. (15)	15.12
Lattice C=16, approx. (13)	15.30
Lattice C=16, approx. (15)	15.27
Lattice m=16	15.52
Lattice m=32	15.26
Lattice m=128	14.84

Ablation. We ablate key components of the Lattice to evaluate their individual contributions to the overall performance. The results in Table 2 (evaluated on 110M parameter models trained on FineWeb-Edu) underscore that both the orthogonal state recurrence (5) and the normalization (9) are essential to the model's overall performance. Furthermore, our analysis reveals that the explicit forget gate's impact on the overall performance is negligible, indicating that the normalized projection introduced in (9) inherently acts as an effective built-in forgetting mechanism. Remarkably, Lattice configured with only a quarter of the memory slots ($m = 16$) outperforms TTT with $m = d = 64$. This demonstrates that *the proposed orthogonal update mechanism utilizes the fixed memory capacity significantly more efficiently*. Extended ablation studies on memory scaling and normalization strategies (per-token vs. per-chunk) are provided in Appendix D.

5 Conclusion

This work introduces a novel recurrent neural network mechanism for efficiently compressing streaming context into a fixed-size, matrix-valued memory. By framing this compression as an online optimization problem, we derived a memory-update rule from a single gradient descent step, resulting in a state- and input-dependent recurrence that is both interpretable and expressive. A core feature of this mechanism is Orthogonal State Recurrence (OSR), wherein each memory slot is updated exclusively with information orthogonal to its current state, ensuring non-redundant storage. Furthermore, this update is complemented by a normalization step that effectively implements a data- and state-dependent gating mechanism, theoretically grounded in a Riemannian optimization perspective. Empirically, Lattice achieves superior memory efficiency and strong performance across language modeling and associative recall tasks, while maintaining subquadratic scaling and a hardware-efficient parallel implementation. Lattice thus offers a promising and scalable alternative to Transformers for efficient pre-training.

REFERENCES

P-A Absil, Robert Mahony, and Rodolphe Sepulchre. Optimization algorithms on matrix manifolds. In *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2009.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

Anurag Arnab, Mostafa Dehghani, Georg Heigold, Chen Sun, Mario Lučić, and Cordelia Schmid. Vivit: A video vision transformer. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6836–6846, 2021.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. *arXiv preprint arXiv:2312.04927*, 2023.

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.

Jimmy Lei Ba. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Maximilian Beck, Korbinian Pöppel, Markus Spanring, Andreas Auer, Oleksandra Prudnikova, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. xLSTM: Extended long short-term memory. *arXiv preprint arXiv:2405.04517*, 2024.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024.

Ali Behrouz, Zeman Li, Praneeth Kacham, Majid Daliri, Yuan Deng, Peilin Zhong, Meisam Razaviyayn, and Vahab Mirrokni. Atlas: Learning to optimally memorize the context at test time. *arXiv preprint arXiv:2505.23735*, 2025.

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7432–7439, 2020.

David M Blei. Topic models. *Text mining/Chapman and Hall/CRC*, 2009.

David M Blei. Probabilistic topic models. *Communications of the ACM*, 55(4):77–84, 2012.

Guy E Blelloch. Prefix sums and their applications. 1990.

Silvere Bonnabel. Stochastic gradient descent on riemannian manifolds. *IEEE Transactions on Automatic Control*, 58(9):2217–2229, 2013.

Nicolas Boumal. *An introduction to optimization on smooth manifolds*. Cambridge University Press, 2023.

Beidi Chen, Tri Dao, Eric Winsor, Zhao Song, Atri Rudra, and Christopher Ré. Scatterbrain: Unifying sparse and low-rank attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Can Chen, Xi Chen, Chen Ma, Zixuan Liu, and Xue Liu. Gradient-based bi-level optimization for deep learning: A survey. *arXiv preprint arXiv:2207.11719*, 2022.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Kevin Clark, Kelvin Guu, Ming-Wei Chang, Panupong Pasupat, Geoffrey Hinton, and Mohammad Norouzi. Meta-learning fast weight language models. *arXiv preprint arXiv:2212.02475*, 2022.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning (ICML)*, 2024.

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.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

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

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

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

Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with LSTM recurrent networks. *Journal of machine learning research*, 3(Aug):115–143, 2002.

Tom Goldstein, Brendan O'Donoghue, Simon Setzer, and Richard Baraniuk. Fast alternating direction optimization methods. *SIAM Journal on Imaging Sciences*, 7(3):1588–1623, 2014.

Xavier Gonzalez, Andrew Warrington, Jimmy TH Smith, and Scott W Linderman. Towards scalable and stable parallelization of nonlinear rnns. *arXiv preprint arXiv:2407.19115*, 2024.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

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

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

Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

Albert Gu, Karan Goel, Ankit Gupta, and Christopher Ré. On the parameterization and initialization of diagonal state space models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=yJE7iQSAep.

Han Guo, Songlin Yang, Tarushii Goel, Eric P Xing, Tri Dao, and Yoon Kim. Log-linear attention. *arXiv preprint arXiv:2506.04761*, 2025.

Simon S Haykin. *Adaptive filter theory*. Pearson Education India, 2002.

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

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

Jiaxi Hu, Yongqi Pan, Jusen Du, Disen Lan, Xiaqiang Tang, Qingsong Wen, Yuxuan Liang, and Weigao Sun. Comba: Improving nonlinear rnns with closed-loop control. *arXiv preprint arXiv:2506.02475*, 2025.

Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. Transformer quality in linear time. In *International conference on machine learning*, pp. 9099–9117. PMLR, 2022.

Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Going beyond linear transformers with recurrent fast weight programmers. *Advances in neural information processing systems*, 34:7703–7717, 2021.

Praneeth Kacham, Vahab Mirrokni, and Peilin Zhong. Polysketchformer: Fast transformers via sketching polynomial kernels. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=ghYrfdJfjK.

Mahdi Karami. HiGen: Hierarchical graph generative networks. In *The Twelfth International Conference on Learning Representations*, 2024.

Mahdi Karami and Ali Ghodsi. Orchid: Flexible and data-dependent convolution for sequence modeling. In *Thirty-eighth Conference on Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2402.18508.

Mahdi Karami, Martha White, Dale Schuurmans, and Csaba Szepesvári. Multi-view matrix factorization for linear dynamical system estimation. *Advances in Neural Information Processing Systems*, 30, 2017.

Mahdi Karami, Dale Schuurmans, Jascha Sohl-Dickstein, Laurent Dinh, and Daniel Duckworth. Invertible convolutional flow. *Advances in Neural Information Processing Systems*, 32, 2019.

Mahdi Karami, Ali Behrouz, Praneeth Kacham, and Vahab Mirrokni. Trellis: Learning to compress key-value memory in attention models. *Second Conference on Language Modeling*, 2025a.

Mahdi Karami, Ali Behrouz, Peilin Zhong, Razvan Pascanu, and Vahab Mirrokni. MS-SSM: A multi-scale state space model for efficient sequence modeling. In *Second Conference on Language Modeling*, 2025b.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pp. 5156–5165. PMLR, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Yuhong Li, Tianle Cai, Yi Zhang, Deming Chen, and Debadeepta Dey. What makes convolutional models great on long sequence modeling? *arXiv preprint arXiv:2210.09298*, 2022.

Zeman Li, Ali Behrouz, Yuan Deng, Peilin Zhong, Praneeth Kacham, Mahdi Karami, Meisam Razaviyayn, and Vahab Mirrokni. Tnt: Improving chunkwise training for test-time memorization. *arXiv preprint arXiv:2511.07343*, 2025.

Yi Heng Lim, Qi Zhu, Joshua Selfridge, and Muhammad Firmansyah Kasim. Parallelizing non-linear sequential models over the sequence length. *arXiv preprint arXiv:2309.12252*, 2023.

Zhixuan Lin, Evgenii Nikishin, Xu Owen He, and Aaron Courville. Forgetting transformer: Softmax attention with a forget gate. *arXiv preprint arXiv:2503.02130*, 2025.

Bo Liu, Mao Ye, Stephen Wright, Peter Stone, and Qiang Liu. Bome! bilevel optimization made easy: A simple first-order approach. *Advances in neural information processing systems*, 35: 17248–17262, 2022.

Bo Liu, Rui Wang, Lemeng Wu, Yihao Feng, Peter Stone, and Qiang Liu. Longhorn: State space models are amortized online learners. *arXiv preprint arXiv:2407.14207*, 2024.

Hanbaek Lyu, Deanna Needell, and Laura Balzano. Online matrix factorization for markovian data and applications to network dictionary learning. *Journal of Machine Learning Research*, 21(251): 1–49, 2020.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th annual international conference on machine learning*, pp. 689–696, 2009.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11(1), 2010.

Harsh Mehta, Ankit Gupta, Ashok Cutkosky, and Behnam Neyshabur. Long range language modeling via gated state spaces. *arXiv preprint arXiv:2206.13947*, 2022.

William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. *arXiv preprint arXiv:2404.08819*, 2024.

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

Antonio Orvieto, Samuel L Smith, Albert Gu, Anushan Fernando, Caglar Gulcehre, Razvan Pascanu, and Soham De. Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning*, pp. 26670–26698. PMLR, 2023.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin A Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024.

Bo Peng, Eric Alcaide, Quentin Gregory Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Nguyen Chung, Leon Derczynski, Xingjian Du, Matteo Grella, Kranthi Kiran GV, Xuzheng He, Haowen Hou, Przemyslaw Kazienko, Jan Kocon, Jiaming Kong, Bartłomiej Koptyra, Hayden Lau, Jiaju Lin, Krishna Sri Ipsit Mantri, Ferdinand Mom, Atsushi Saito, Guangyu Song, Xiangru Tang, Johan S. Wind, Stanisław Woźniak, Zhenyuan Zhang, Qinghua Zhou, Jian Zhu, and Rui-Jie Zhu. RWKV: Reinventing RNNs for the transformer era. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=7SaXczaBpG.

Bo Peng, Daniel Goldstein, Quentin Anthony, Alon Albalak, Eric Alcaide, Stella Biderman, Eugene Cheah, Xingjian Du, Teddy Ferdinan, Haowen Hou, et al. Eagle and finch: RWKV with matrix-valued states and dynamic recurrence. *arXiv preprint arXiv:2404.05892*, 2024.

Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Xingjian Du, Haowen Hou, Jiaju Lin, Jiaxing Liu, Janna Lu, William Merrill, et al. Rwkv-7" goose" with expressive dynamic state evolution. *arXiv preprint arXiv:2503.14456*, 2025.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. *International Conference on Machine Learning*, 2023.

DL Prados and SC Kak. Neural network capacity using delta rule. *Electronics Letters*, 25(3):197–199, 1989.

Zhen Qin, Xiaodong Han, Weixuan Sun, Dongxu Li, Lingpeng Kong, Nick Barnes, and Yiran Zhong. The devil in linear transformer. *arXiv preprint arXiv:2210.10340*, 2022.

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

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

David W Romero, Anna Kuzina, Erik J Bekkers, Jakub Mikolaj Tomczak, and Mark Hoogendoorn. Ckconv: Continuous kernel convolution for sequential data. In *International Conference on Learning Representations*, 2021.

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

Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, pp. 9355–9366. PMLR, 2021.

JH Schmidhuber. Learning to control fast-weight memories: An alternative to recurrent nets. *Neural Computation*, 1992.

Julien Siems, Timur Carstensen, Arber Zela, Frank Hutter, Massimiliano Pontil, and Riccardo Grazzi. Deltaproduct: Improving state-tracking in linear rnns via householder products. *arXiv preprint arXiv:2502.10297*, 2025.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention. *arXiv preprint arXiv:2406.02542*, 2024.

Jimmy T.H. Smith, Andrew Warrington, and Scott Linderman. Simplified state space layers for sequence modeling. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Ai8Hw3AXqks.

Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, 46(1-2):159–216, 1990.

Vipin Srivastava, Suchitra Sampath, and David J Parker. Overcoming catastrophic interference in connectionist networks using gram-schmidt orthogonalization. *PloS one*, 9(9):e105619, 2014.

Gilbert Strang. *Linear algebra and its applications*. 2000.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *International conference on machine learning*, pp. 9229–9248. PMLR, 2020.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, et al. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

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

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. *arXiv preprint arXiv:1811.00937*, 2018.

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.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, 2022.

Sebastian Thrun and Lorien Pratt. Learning to learn: Introduction and overview. In *Learning to learn*, pp. 3–17. Springer, 1998.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017a.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. volume 30, 2017b.

Christoph Von Der Malsburg. The correlation theory of brain function. In *Models of neural networks: Temporal aspects of coding and information processing in biological systems*, pp. 95–119. Springer, 1994.

Johannes von Oswald, Nino Scherrer, Seijin Kobayashi, Luca Versari, Songlin Yang, Maximilian Schlegel, Kaitlin Maile, Yanick Schimpf, Oliver Sieberling, Alexander Meulemans, et al. Mesanet: Sequence modeling by locally optimal test-time training. *arXiv preprint arXiv:2506.05233*, 2025.

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

Sinong Wang, Belinda Z Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. In *Neurocomputing: foundations of research*, pp. 123–134. 1988.

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

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *Forty-first International Conference on Machine Learning*, 2024b. URL https://openreview.net/forum?id=ia5XvxFUJT.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024c. URL https://openreview.net/forum?id=y8Rm4VNRPH.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. *Advances in neural information processing systems*, 32, 2019.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

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

Yu Zhang, Songlin Yang, Ruijie Zhu, Yue Zhang, Leyang Cui, Yiqiao Wang, Bolun Wang, Freda Shi, Bailin Wang, Wei Bi, et al. Gated slot attention for efficient linear-time sequence modeling. *arXiv preprint arXiv:2409.07146*, 2024.

Broader Impacts

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

A Related Works

Fast Weight Programmers and Test-Time Training. The two-stage learning process adopted in our work draws inspiration from the concept of Fast Weight Programmers (FWPs) (Schmidhuber, 1992; Schlag et al., 2021) where a "slow" network dynamically updates the parameters of a "fast" network. In our framework, the compression layer in the inner loop can be seen as the fast network, with its memory states, $\mathbf{S}_t$, acting as "fast weights" that are rapidly adapted to the evolving contextual information. The outer loop, conversely, learns the generalizable parameters of the slow neural network, optimized across the entire training dataset. The continual reprogramming of fast network weights by slow models (Irie et al., 2021; Clark et al., 2022) is broadly recognized as Fast Weight Programming, also referred to as synaptic modulation (Von Der Malsburg, 1994) or input-dependent parameterization (Karami et al., 2019; Gu & Dao, 2023; Karami & Ghodsi, 2024), a technique known to enhance model expressiveness. In our architecture, the parameterization of the linear projections by the slow network facilitates this fast adaptation within the inner loop. Similarly, *Test-Time Training* (Sun et al., 2020; 2024; Behrouz et al., 2024; von Oswald et al., 2025) is a paradigm where a model adapts to each test instance by optimizing a self-supervised objective before making predictions. Our compression layer effectively implements a form of test-time training by dynamically updating its state based on the contextual information of the input sequence during inference. In contrast to the aforementioned works, our approach introduces an explicit learning mechanism for the "fast" compression layer, leading to an interpretable update rule for its internal states that optimally compresses the latest token into memory at test time.

Test-time training for sequence models have recently been formalized through the lens of online optimization (Sun et al., 2024). The Test Time Regression (TTR) (Wang et al., 2025) and Longhorn (Liu et al., 2024) propose linear recurrences with state-independent gating. Works such as Karami et al. (2025a); von Oswald et al. (2025); Zhang et al. (2025) fall under this category. They have demonstrated that deriving recurrent update rules from the online optimization of a regression objective can yield powerful sequence models.

Adaptive Filters. Classical adaptive filtering algorithms (Haykin, 2002) iteratively update their weights to minimize prediction error while efficiently adapting to streaming, non-stationary data. These methods share core principles with the online learning and dynamic memory updates employed in our work. In particular, the gradient descent-based update rules we adopted for memory adaptation are closely related to the Least Mean Squares (LMS) algorithm—also known as the Widrow-Hoff algorithm (Widrow & Hoff, 1988)—which updates weights using the instantaneous gradient of the squared error. Furthermore, variations such as Normalized Least Mean Squares (NLMS), which involves a normalized step size for improved convergence, and Leaky LMS, which incorporates a leakage factor used to prevent unbounded growth of filter weights, find parallels in our use of normalization mechanisms to stabilize memory update (Equation 9) and state decay (Equation 10). While these adaptive filtering methods rely on linear weight updates, our approach introduces a non-linear memory update rule that incorporates only the non-redundant components of the new token.

Matrix Factorization Matrix factorization and dictionary learning are classical representation learning techniques that aim to extract essential features from complex data by approximating it as a linear combination of a reduced set of basis vectors, also known as dictionary atoms. This concept is also conceptually related to topic modeling, where the objective is to extract important features (topics) from a complex dataset to obtain a reduced representation (Blei, 2009; 2012). Mairal et al. (2010) proposed an online optimization algorithm for structured matrix factorization and sparse coding for an i.i.d. stream of data, which efficiently scales to large datasets. Subsequently, Lyu et al. (2020) extended this work by proving the convergence of such an online algorithm in non-i.i.d. settings, where the sequential data form a Markov chain. In a related area, Karami et al. (2017) formulated the identification of SSMs (a.k.a. linear dynamical systems) as a multi-view matrix factorization problem and proposed a convex optimizer for its solution. In contrast to the online matrix factorization in Mairal et al. (2009), which employs a model-free method to learn the latent coefficients (codes) and leverages block coordinate descent for optimization, our method formulates the memory update as a fast internal optimization procedure. We incorporate a simple encoding layer to generate the latent representation, $\boldsymbol{k}_t$, and integrate it into a larger deep neural network training procedure.

Table 3: Comparison of the objectives and their corresponding online updates for the proposed and existing RNNs. We include: Linear-Attention(LA) (Katharopoulos et al., 2020), Mamba2 (Dao & Gu, 2024) and DeltaNet (Schlag et al., 2021; Yang et al., 2024c), GLA (Yang et al., 2024b), RWKV-6/7 (Peng et al., 2024; 2025), Gated-DeltaNet (Yang et al., 2024a), and TTT (Sun et al., 2024). Note that the effective recurrent update of the compression layers after re-scaling becomes online GD on a Riemannian manifold: $\mathbf{S}_t = \mathbf{1}\,\boldsymbol{\beta}_t^\top \odot \big(\mathbf{S}_{t-1} + \Delta\mathbf{S}_t\big)$ (eq. 9). LA is equivalent to online GD with a fixed step size ($\gamma_t = 1$); however, input-dependent step sizes are frequently used in recent RNNs (Orvieto et al., 2023; Qin et al., 2024; Gu & Dao, 2023). Mamba2 and Gated-DeltaNet employ a forgetting gate, which is equivalent to online GD with L2 regularization of strength λ_t. In Mamba2, the forget gate is controlled by $\mu_t = 1-\lambda_t$, and the reparameterization for the forget gate and step size of Gated-DeltaNet is discussed in (Wang et al., 2025). In RWKV-7, the regularizer is column-wise: $\frac{1}{2}\|\mathbf{S}_t^\top\|^2_{\mathrm{diag}(\boldsymbol{\lambda}_t)} := \frac{1}{2}\,\mathrm{Tr}(\mathbf{S}_t^\top\,\mathrm{diag}(\boldsymbol{\lambda}_t)\mathbf{S}_t)$, resulting in a diagonal-plus-low-rank transition matrix. In comparison, Table 4 in Liu et al. (2024) list a closed-form solution for online learning in linear layers. Here, $\times_2$ denotes vector-tensor product defined as $\big[\mathbf{J}_1,\,\ldots,\mathbf{J}_m\big]\times_2 \boldsymbol{e} = \big[\mathbf{J}_1\boldsymbol{e},\,\ldots,\mathbf{J}_m\boldsymbol{e}\big]$.

Method	**Objective $\mathcal{L}_t$**	**Online Gradient Descent Update**
Linear-Attention	$-\langle \mathbf{S}_t\boldsymbol{k}_t, \boldsymbol{v}_t\rangle$	$\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t\boldsymbol{k}_t^\top$
Mamba2	$-\langle \mathbf{S}_t\boldsymbol{k}_t, \boldsymbol{v}_t\rangle + \frac{\lambda_t}{2}\|\mathbf{S}_t\|_2^2$	$\mathbf{S}_t = \mu_t\mathbf{S}_{t-1} + \boldsymbol{v}_t\boldsymbol{k}_t^\top$
RWKV-6 & GLA	$-\langle \mathbf{S}_t\boldsymbol{k}_t, \boldsymbol{v}_t\rangle + \frac{\boldsymbol{\lambda}_t}{2}\|\mathbf{S}_t\|_2^2$	$\mathbf{S}_t = \mathrm{diag}(\boldsymbol{\mu}_t)\,\mathbf{S}_{t-1} + \boldsymbol{v}_t\boldsymbol{k}_t^\top$
DeltaNet	$\Big\|\mathbf{S}_t\boldsymbol{k}_t - \boldsymbol{v}_t\Big\|^2$	$\mathbf{S}_t = \mathbf{S}_{t-1}(\mathbf{I} - \gamma_t\boldsymbol{k}_t\boldsymbol{k}_t^T) + \gamma_t\boldsymbol{v}_t\boldsymbol{k}_t^T$
Gated-DeltaNet	$\Big\|\mathbf{S}_t\boldsymbol{k}_t - \boldsymbol{v}_t\Big\|^2 + \frac{\lambda_t}{2}\|\mathbf{S}_t\|_2^2$	$\mathbf{S}_t = \mu_t\mathbf{S}_{t-1}(\mathbf{I} - \gamma_t\boldsymbol{k}_t\boldsymbol{k}_t^T) + \gamma_t\boldsymbol{v}_t\boldsymbol{k}_t^T$
RWKV-7	$\Big\|\mathbf{S}_t\boldsymbol{k}_t - \boldsymbol{v}_t\Big\|^2 + \frac{1}{2}\|\mathbf{S}_t^\top\|^2_{\mathrm{diag}(\boldsymbol{\lambda}_t)}$	$\mathbf{S}_t = \mathbf{S}_{t-1}(\mathrm{diag}(\boldsymbol{\mu}_t) - \gamma_t\boldsymbol{k}_t\boldsymbol{k}_t^T) + \gamma_t\boldsymbol{v}_t\boldsymbol{k}_t^T$
TTT	$\Big\|\phi(\mathbf{S}_t\boldsymbol{k}_t) - \boldsymbol{v}_t\Big\|^2$	$\mathbf{S}_t = \mathbf{S}_{t-1} - \gamma_t\Big(\frac{\mathbf{P}(\boldsymbol{z}_t)}{\|\boldsymbol{z}_t\|}\boldsymbol{e}_t\Big)\boldsymbol{k}_t^\top$
Lattice (Dec) (4)	$\Big\|\phi(\mathbf{S}_t)\boldsymbol{k}_t - \boldsymbol{v}_t\Big\|^2$	$\mathbf{S}_t = \mathbf{1}\,\boldsymbol{\beta}_t^\top \odot (\mathbf{S}_{t-1} - \gamma_t\big(\Big[\frac{\mathbf{P}(\boldsymbol{s}_1)}{\|\mathbf{s}_1\|},\ldots,\frac{\mathbf{P}(\boldsymbol{s}_m)}{\|\mathbf{s}_m\|}\Big]\times_2 \boldsymbol{e}_t\big)\odot\boldsymbol{k}_t^\top)$
Lattice (Sim) (6)	$-\langle \phi(\mathbf{S}_t)^\top\boldsymbol{v}_t,\ \boldsymbol{k}_t\rangle$	$\mathbf{S}_t = \mathbf{1}\,\boldsymbol{\beta}_t^\top \odot (\mathbf{S}_{t-1} + \gamma_t\big(\Big[\frac{\mathbf{P}(\boldsymbol{s}_1)}{\|\mathbf{s}_1\|},\ldots,\frac{\mathbf{P}(\boldsymbol{s}_m)}{\|\mathbf{s}_m\|}\Big]\times_2 \boldsymbol{v}_t\big)\odot\boldsymbol{k}_t^\top)$
Lattice (Enc) (22)	$\Big\|\phi(\mathbf{S}_t)^\top\boldsymbol{v}_t - \boldsymbol{k}_t\Big\|^2$	$\mathbf{S}_t = \mathbf{1}\,\boldsymbol{\beta}_t^\top \odot (\mathbf{S}_{t-1} - \gamma_t\big(\Big[\frac{\mathbf{P}(\boldsymbol{s}_1)}{\|\mathbf{s}_1\|},\ldots,\frac{\mathbf{P}(\boldsymbol{s}_m)}{\|\mathbf{s}_m\|}\Big]\times_2 \boldsymbol{v}_t\big)\odot\boldsymbol{e}_t^\top)$

Recent Advancements in RNNs. More broadly, the field is witnessing a resurgence of interest in RNN research, with many new architectures emerging as viable alternatives to Transformers. Works such as Sun et al. (2023); Lin et al. (2025); Merrill et al. (2024); Kacham et al. (2024); Peng et al. (2023; 2025); Guo et al. (2025); Du et al. (2025); Siems et al. (2025); Hu et al. (2025); Karami et al. (2025b); Behrouz et al. (2025); Zhang et al. (2025); Tandon et al. (2025) tackle the limitations of RNNs, introducing new mechanisms to improve long-range dependency, handling scalability, or efficient parallelizability.

B Discussion

Remark B.1 (Delta Rule) *Removing the non-linearity $\phi(\cdot)$ from the compression layer simplifies the online gradient descent update rule in (3) to*

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \gamma_t(\mathbf{S}_{t-1}\boldsymbol{k}_t - \boldsymbol{v}_t)\boldsymbol{k}_t^\top = \mathbf{S}_{t-1}(\mathbf{I} - \gamma_t\boldsymbol{k}_t\boldsymbol{k}_t^\top) + \gamma_t\boldsymbol{v}_t\boldsymbol{k}_t^\top \tag{16}$$

This linear update rule recovers the delta rule (Widrow & Hoff, 1988), known for its higher memory capacity (Prados & Kak, 1989) and has been demonstrated as an effective form of linear recurrence, particularly in associative recall tasks (Schlag et al., 2021; Yang et al., 2024c). Similar to linear transformers, the second term writes into memory via the outer product $\boldsymbol{v}_t\boldsymbol{k}_t^\top$, while the first term implements a forgetting mechanism, controlled by the new key $\boldsymbol{k}_t$, to remove old information from memory. Here, we propose a more efficient update rule based on the nonlinear interactions between the memory and the non-redundant information of the new keys.

Remark B.2 (Layer Normalization) *In the proposed compression layers, normalization was applied to each state column. Alternatively, Sun et al. (2024) proposed applying normalization on the output of the decoding layer.*[7] *In this case, the decoding function becomes: $\hat{\boldsymbol{v}}_t = g(\boldsymbol{k}_t; \mathbf{S}_t) = \phi(\mathbf{S}_t\boldsymbol{k}_t)$.*

[7]While the general formulation of TTT (Sun et al., 2024) applies a non-linearity to $\boldsymbol{z}_t$, their implementation specifically utilizes normalization.

This formulation is analogous to applying layer normalization *as commonly used in deep neural networks. As before, we can simplify the gradient $\nabla_{\mathbf{S}}\mathcal{L}_t$ to derive an interpretable update rule. Specifically, let $\phi(\boldsymbol{z}_t) = \frac{\boldsymbol{z}_t}{\|\boldsymbol{z}_t\|}$, where $\boldsymbol{z}_t := \mathbf{S}_{t-1}\boldsymbol{k}_t$, and define the reconstruction error as $\boldsymbol{e}_t := \hat{\boldsymbol{v}}_t - \boldsymbol{v}_t$. Applying the chain rule, we obtain:*

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{S}} = \big(\mathbf{J}_\phi(\boldsymbol{z}_t)\boldsymbol{e}_t\big)\,\boldsymbol{k}_t^\top, \text{ where } \mathbf{J}_\phi(\boldsymbol{z}_t) = \frac{\mathbf{P}(\boldsymbol{z}_t)}{\|\boldsymbol{z}_t\|} = \frac{1}{\|\boldsymbol{z}_t\|}\left(\mathbf{I} - \frac{\boldsymbol{z}_t\boldsymbol{z}_t^\top}{\|\boldsymbol{z}_t\|^2}\right)$$

Subsequently, the gradient descent update follows a nonlinear recurrence:

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \gamma_t\Big(\frac{\mathbf{P}(\boldsymbol{z}_t)}{\|\boldsymbol{z}_t\|}\boldsymbol{e}_t\Big)\,\boldsymbol{k}_t^\top, \tag{17}$$

This nonlinear state recurrence incorporates an outer-product correction based on the projection of the reconstruction error $\boldsymbol{e}_t$ onto the orthogonal complement space of $\hat{\boldsymbol{v}}_t$.

The concept of normalizing the state vectors in our compression model, as described in §3.1, shares similarities with weight normalization techniques used in deep learning literature (Salimans & Kingma, 2016). Furthermore, the two interpretations presented above—applying normalization to the output of the decoding layer versus normalizing the state vectors—offer insights into the rationale behind different normalization schemes commonly used in deep learning, such as weight normalization and layer normalization (Ba, 2016). Each normalization method plays a distinct role in stabilizing training and improving generalization.

Remark B.3 (Parallelization of Lattice **vs.** TTT**)** *In* TTT*, the non-linearity $\phi(\cdot)$ is applied to the activation output. This yields a simple rank-one gradient: $\nabla_S\mathcal{L}_t = [J_\phi(\boldsymbol{z}_t)^\top(\phi(\boldsymbol{z}_t) - \boldsymbol{v}_t)]\boldsymbol{k}_t^\top = \boldsymbol{u}_t\boldsymbol{k}_t^\top$, where $\boldsymbol{u}_t = J_\phi(\boldsymbol{z}_t)^\top\boldsymbol{e}_t$ is a preconditioned error vector. Because this gradient forms a standard outer product, it linearizes within a chunk into an additive recurrence form: $\mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{u}_t\boldsymbol{k}_t^\top$, which is similar to standard Linear Attention, making parallelization mathematically straightforward.*

Conversely, Lattice *applies the non-linearity ϕ directly to the state matrix $\mathbf{S}$ itself, resulting in the column-wise gradient:*

$$\nabla_{\boldsymbol{s}_i}\mathcal{L}_t = k_{i,t}J_\phi(\boldsymbol{s}_i)\boldsymbol{e}_t$$

This does not directly factorize to the above standard rank-one outer product, making a chunk-wise matrix form non-trivial. To overcome this, we derived a specialized strategy relying on the specific geometric properties of the orthogonal projection. In §3.4.1, we derive two approximations of the gradient: I) zeroth-order: approximating the gradient as a constant function of $\mathbf{S}_{t-1}$ in equation 14, II) first-order: approximating it as an affine function of $\mathbf{S}_{t-1}$ in (15), preserving the first-order dependency of the gradient on $\mathbf{S}_{t-1}$. We simplified these into hardware-efficient forms in Appendix C.1.

B.1 Novelties of Lattice Relative to TTT

In this section we briefly overview the main advantages of the proposed method compared to TTT (Sun et al., 2024):

- **Mechanism and Interpretation**: While Lattice shares the high-level perspective of framing sequence modeling as an online optimization problem, it fundamentally diverges by applying its non-linearity directly to the memory states rather than the activations. Rooted in subspace and dictionary learning, this basis normalization resolves scale ambiguity by decoupling a feature's semantic direction ($\boldsymbol{s}_i$) from its magnitude or importance. Moreover, the interpretable Orthogonal State Recurrence (OSR) is a direct mathematical consequence of this state normalization in the decoding layer (4). The necessity of orthogonalization for stable memory is supported in *neuroscience-inpired* memory management methods, for instance, Srivastava et al. (2014) demonstrate that applying Gram-Schmidt orthogonalization to incoming inputs prevents catastrophic interference in connectionist networks. However, while their approach requires each new pattern to be orthogonal to all previous ones, Lattice orthogonalizes the update direction relative only to the current state of each specific memory slot. This offers a highly scalable mechanism to selectively integrate novel information without requiring full historical orthogonalization.

- **Geometric Gating Mechanism**: The normalization step in equation 9 is theoretically grounded in Riemannian optimization on a spherical manifold, not just an ad-hoc stability tweak. This constraint *naturally induces a data- and state-dependent gating mechanism (acting as dynamic forget/input gates)*. This contrasts with TTT, which does not explicitly address the vanishing or exploding state norm problem.
- **Specialized Parallelization Strategy**: Linearizing our specific recurrence for chunk-wise computation requires a specialized strategy that is distinct from the TTT's standard chunk-wise approach. In §3.4.1, we derived a zeroth- and first-order chunk-wise parallelization strategy for this state-dependent nonlinearity.
- **Empirical Significance**: These theoretical derivations translate into substantial *empirical advantages*. Most notably, our scaling studies demonstrate a $4\times$ gain in memory efficiency: Lattice utilizing only a quarter of the memory slots ($m = 16$) outperforms TTT and DeltaNet with ($m = d = 64$). This architectural efficiency is critical for long-context inference. Furthermore, Lattice's relative performance gains over strong baselines widen as context scales from 512 to 16k tokens. Finally, our MQAR results prove Lattice's superior capability to store and retrieve dense information—an advantage that becomes increasingly prominent in longer sequences—whereas models like Gated-DeltaNet and GLA fail to effectively learn this task across most model dimensions.

C Detailed Derivations and Model Variants

C.1 Parallel and Hardware Efficient Form

Various methods have been explored to enable parallel evaluation of non-linear RNNs. One strategy, as proposed by Lim et al. (2023); Gonzalez et al. (2024), involves casting inference as finding the solution to a fixed-point equation, thereby achieving parallelism. In a different approach, Sun et al. (2024) introduced a parallel chunk-wise solution using a gradient approximation. This method splits a sequence into non-overlapping chunks and utilizes the state at the beginning of each chunk to approximate the gradients for the entire chunk in parallel. Following this technique, we let $\mathbf{S}_{t'}$ represent the state at the beginning of the chunk (i.e., the final state from the preceding chunk), where $t' = C\lfloor (t-1)/C \rfloor$ with C denoting the chunk size. The gradient is then approximated as $\nabla_S \mathcal{L}(\mathbf{S}_{t'}, \boldsymbol{v}_t, \boldsymbol{k}_t)$.This approximation linearizes the nonlinear recurrence (12) as:

$$\mathbf{S}_t = \mathbf{G}_t \odot \mathbf{S}_{t-1} + \mathbf{1}(\tilde{\boldsymbol{h}}_t \odot \hat{\boldsymbol{k}}_t)^\top \odot \mathbf{S}_{t'} - \boldsymbol{h}_t \hat{\boldsymbol{k}}_t^\top, \tag{18}$$

where $\tilde{\boldsymbol{h}}_t = \mathbf{S}_{t'}^\top \boldsymbol{h}_t$. Now, for the time steps $t = bC+\tau$ in the b-th block, let $\mathbf{X}^b = \boldsymbol{x}[bC+1 : (b+1)C]$ denote the stacked input into the chunk-wise matrices and $\mathbf{X}_\tau^b = \boldsymbol{x}[bC+\tau]$ (similarly for other vectors such as $\boldsymbol{q}, \boldsymbol{h}, \boldsymbol{k}$), and define the local cumulative product of decay factors as $\boldsymbol{a}_\tau^b = \prod_{i=bC+1}^{bC+\tau} \boldsymbol{\beta}_i \mu_i$ and $\boldsymbol{A}^b = [\boldsymbol{a}_\tau^b]_{\tau=1}^C$. We also define a block lower triangular tensor $\boldsymbol{\Omega}^b \in \mathbb{R}^{C\times C\times m}$ with components $\boldsymbol{\Omega}^b_{j,i,:} = \frac{\boldsymbol{a}_j^b}{\boldsymbol{a}_i^b} \mathbb{I}_{i\le j}$ that are segmented cumulative product over the sub-block steps i to j ($1 \le i \le j \le C$). Here $\mathbb{I}_{i\le j}$ is the indicator function (equal to 1 if $i \le j$ and 0 otherwise), the division $\frac{\boldsymbol{a}_j^b}{\boldsymbol{a}_i^b}$ is performed element-wise. Therefore, the recurrence equation 13 can be expressed at the chunk-level as:

$$\begin{aligned} \mathbf{S}_b &= \mathbf{1}(\boldsymbol{a}_C^b + \boldsymbol{f}^b)^\top \odot \mathbf{S}_{b-1} - \boldsymbol{H}^{b\top}(\hat{\boldsymbol{K}}^b \odot \boldsymbol{\Omega}^b_{C,:,:}) \\ \boldsymbol{Y}^b &= \big(\boldsymbol{Q}^b \odot (\boldsymbol{A}^b + \boldsymbol{F}^b)\big)\mathbf{S}_{b-1}^\top - \boldsymbol{P}^b \boldsymbol{H}^b \end{aligned} \tag{19}$$

where $\boldsymbol{f}^b = \mathrm{diag}[\tilde{\boldsymbol{H}}^{b\top}(\hat{\boldsymbol{K}}^b \odot \boldsymbol{\Omega}^b_{C,:,:})] \in \mathbb{R}^m$, and $\boldsymbol{F}^b \in \mathbb{R}^{C\times m}$ is a matrix with entries $\boldsymbol{F}^b_{jn} = \sum_{\tau=1}^C (\tilde{\boldsymbol{H}}^b \odot \hat{\boldsymbol{K}}^b)_{\tau n} \boldsymbol{\Omega}^b_{j\tau n}\ \forall\, 0 < j \le C,\ 0 < n \le m$. Both of these can be computed using *tensor contraction* or Einstein summation operation as

$$\boldsymbol{f}^b = \sum_{\tau=1}^{C} \tilde{\boldsymbol{H}}^b_{\tau:} \odot (\hat{\boldsymbol{K}}^b \odot \boldsymbol{\Omega}^b_{C::})_{\tau:} = \texttt{einsum("C m, C m -> m", } \tilde{\boldsymbol{H}}^b, (\hat{\boldsymbol{K}}^b \odot \boldsymbol{\Omega}^b_{C::}))$$

$$\boldsymbol{F}^b = \texttt{einsum("C B m, B m -> C m", } \boldsymbol{\Omega}^b, (\tilde{\boldsymbol{H}}^b \odot \hat{\boldsymbol{K}}^b))).$$

Here, $\boldsymbol{P}^b \in \mathbb{R}^{C\times C}$ is a lower triangular matrix with $\boldsymbol{P}^b_{ij} = \sum_{k=1}^{m} \boldsymbol{Q}^b_{ik}\,\hat{\boldsymbol{K}}^b_{jk}\,\boldsymbol{\Omega}^b_{ijk}$. A `matmul`-optimal computation for $\boldsymbol{P}^b$ is presented in Zhang et al. (2024) using a sub-tiling technique. However, a closer inspection of (12) reveals that the recurrence can be simplified by linearizing only the $\hat{\boldsymbol{h}}_t$ term, leading to:

$$\mathbf{S}_t = \hat{\mathbf{G}}_t \odot \mathbf{S}_{t-1} - \boldsymbol{h}_t \hat{\boldsymbol{k}}_t^\top, \quad \text{where } \hat{\mathbf{G}}_t = \mathbf{1}(\mu_t \boldsymbol{\beta}_t + \tilde{\boldsymbol{h}}_t \odot \hat{\boldsymbol{k}}_t)^\top. \tag{20}$$

Here, $\hat{\mathbf{G}}_t$ is parameterized as a rank-one outer product, and hence this intra-chunk update can be computed efficiently using the parallel form of gated linear attention (GLA) presented in Zhang et al. (2024).

Computing $\boldsymbol{\beta}$. For the normalization factor, $\boldsymbol{\beta}_\tau$, in equation 9, we need to compute $\|\Delta \mathbf{s}_{i,\tau}\|^2 \;\forall\, 0 < i \le m,\; 0 < \tau \le C$ which require materializing $\Delta \boldsymbol{S}_\tau \in \mathbb{R}^{d\times m}\;\forall\, 0 < \tau \le C$, that is memory and compute inefficient. However, we can see that these norms can be efficiently computed using matrix multiplication:

$$\begin{aligned}
\Delta \boldsymbol{S}_\tau &= -\gamma_t \nabla_S \mathcal{L}(\mathbf{S}_{t-1}, \boldsymbol{v}_t, \boldsymbol{k}_t) = \gamma_t \big(\mathbf{1}(\tilde{\boldsymbol{h}}_t \odot \tilde{\boldsymbol{k}}_t)^\top \odot \mathbf{S}_0 - \boldsymbol{h}_t \tilde{\boldsymbol{k}}_t^\top\big) \\
\boldsymbol{\beta}_\tau &= \big(\mu_\tau^2 + \operatorname{diag}[\Delta \boldsymbol{S}_\tau^\top\, \Delta \boldsymbol{S}_\tau]\big)^{-\frac{1}{2}} \\
\boldsymbol{d}_{S_\tau} := \operatorname{diag}[\Delta \boldsymbol{S}_\tau^\top\, \Delta \boldsymbol{S}_\tau] &= \gamma_t^2 \big(\|\boldsymbol{h}_t\|^2 \tilde{\boldsymbol{k}}_t^2 - \operatorname{diag}[\boldsymbol{S}_0^\top\, \boldsymbol{S}_0] \odot (\tilde{\boldsymbol{h}}_t \odot \tilde{\boldsymbol{k}}_t)^2\big)
\end{aligned}$$

where $\tilde{\boldsymbol{k}}_t = \gamma_t \boldsymbol{k}_t$. We can also compute $\boldsymbol{d}_{S_\tau}\;\forall\; 0 < \tau \le C$ for the entire chunk by matrix multiplications as

$$\begin{aligned}
\boldsymbol{D}_S := [\boldsymbol{d}_{S_1}; \ldots; \boldsymbol{d}_{S_C}] &= \gamma_t^2 \big(\operatorname{diag}[\boldsymbol{H}\boldsymbol{H}^\top] \odot \tilde{\boldsymbol{K}}^2 - \operatorname{diag}[\boldsymbol{S}_0^\top\, \boldsymbol{S}_0] \odot (\tilde{\boldsymbol{H}} \odot \tilde{\boldsymbol{K}})^2\big) \;\in \mathbb{R}^{C\times m} \\
[\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_C] &= \big([\mu_1^2; \ldots; \mu_C^2]\, \mathbf{1}^\top + \boldsymbol{D}_S\big)^{-\frac{1}{2}}
\end{aligned} \tag{21}$$

where all the squares and square roots are performed element-wise.

This matrix form computes the states only at the end of each block massively save computation and I/O overhead, while enabling efficient use of the `matmul` operations on modern accelerators, a technique used in (Hua et al., 2022; Kacham et al., 2024; Dao & Gu, 2024; Zhang et al., 2024; Sun et al., 2023) for linear RNNs.

C.2 Encoding Layer

Principal Component Analysis (PCA) can be formulated as a linear regression problem, where the data is projected onto a lower-dimensional latent space (Goodfellow et al., 2016, §5.8) Inspired by this regression perspective, we define a encoding layer as: $\hat{\boldsymbol{k}}_t = f(\boldsymbol{v}_t; \mathbf{S}_t) = \phi(\mathbf{S}_t)^\top \boldsymbol{v}_t$ with the corresponding ℓ_2 loss:

$$\mathcal{L}_t = \big\|\phi(\mathbf{S}_t)^\top \boldsymbol{v}_t - \boldsymbol{k}_t\big\|^2, \;\; \mathbf{S}_t \in \mathbb{R}^{d\times m},\; \boldsymbol{v}_t \in \mathbb{R}^d,\; \boldsymbol{k}_t \in \mathbb{R}^m \tag{22}$$

From this, we can derive a closed-form expression for the gradient, resulting in the recurrence:

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \gamma_t \Big(\Big[\tfrac{\mathbf{P}(\boldsymbol{s}_1)}{\|\mathbf{s}_1\|}, \ldots, \tfrac{\mathbf{P}(\boldsymbol{s}_m)}{\|\mathbf{s}_m\|}\Big] \times_2 \boldsymbol{v}_t\Big) \odot \boldsymbol{e}_t^\top \tag{23}$$

General form. The formulations presented in Eqs. (5, 23, and 7) offer principled approaches for designing compression layers in our framework. In general, we refer to this update rule as *Orthogonal State Recurrence (OSR)*, which unifies compression layer into a common framework formulated as follows

$$\{\boldsymbol{y}_t\}_{t=1}^T = \text{Lattice}(\{\boldsymbol{k}_t, \boldsymbol{v}_t, \boldsymbol{q}_t\}_{t=1}^T) := \begin{cases} \mathbf{S}_t = \mathbf{S}_{t-1} - \gamma_t \Big(\Big[\tfrac{\mathbf{P}(\boldsymbol{s}_1)}{\|\mathbf{s}_1\|}, \ldots, \tfrac{\mathbf{P}(\boldsymbol{s}_m)}{\|\mathbf{s}_m\|}\Big] \times_2 \boldsymbol{h}_t\Big) \odot \boldsymbol{c}_t^\top \\ \boldsymbol{y}_t = \mathbf{S}_t \boldsymbol{q}_t \end{cases} \tag{24}$$

Here, the definitions of $\boldsymbol{h}_t$ and $\boldsymbol{c}_t$ vary depending on the specific layer:

$$\begin{cases} \{\boldsymbol{h}_t = \boldsymbol{e}_t, \quad \boldsymbol{c}_t = \boldsymbol{k}_t\}, & \text{Decoding Layer} \\ \{\boldsymbol{h}_t = \boldsymbol{v}_t, \quad \boldsymbol{c}_t = \boldsymbol{e}_t\}, & \text{Encoding Layer} \\ \{\boldsymbol{h}_t = -\boldsymbol{v}_t, \quad \boldsymbol{c}_t = \boldsymbol{k}_t\}, & \text{Similarity Objective} \end{cases}$$

Table 3 provides a summary comparing the online gradient descent-based recurrent corresponding to the proposed compression layers and those of existing RNNs.

C.3 PROOFS

C.3.1 ENCODER LAYER WITH STATE NORMALIZATION

The reconstruction loss in this case is

$$\mathcal{L}_t = \left\| \phi(\mathbf{S}_t)^\top \boldsymbol{v}_t - \boldsymbol{k}_t \right\|^2,$$

where $\mathbf{S}_t \in \mathbb{R}^{d\times m}$ with columns $\boldsymbol{s}_i$ (for $i = 1, \ldots, m$), $\boldsymbol{v}_t \in \mathbb{R}^d$, $\boldsymbol{k}_t \in \mathbb{R}^m$ and $\phi(\mathbf{S}_t) = [\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_m]$ is obtained by normalizing each column of $\mathbf{S}_t$; that is, $\boldsymbol{\phi}_i = \frac{\boldsymbol{s}_i}{\|\boldsymbol{s}_i\|}$. Decomposing the reconstruction error in a per-basis (per-column) form and defining the reconstruction error,

$$\boldsymbol{e}_i := \boldsymbol{\phi}_i^\top \boldsymbol{v}_t - (\boldsymbol{k}_t)_i \; \forall \; i = 1, \ldots, m$$

Then the loss is $\mathcal{L}_t = \sum_{i=1}^m \boldsymbol{e}_i^2$. By this decomposition, we can derive the gradient with respect to each column $\boldsymbol{s}_i$ separately:

$$\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{s}_i} = 2\, \boldsymbol{e}_i \, \frac{\partial \boldsymbol{e}_i}{\partial \boldsymbol{s}_i}.$$

The Jacobian of the normalized vector $\boldsymbol{\phi}_i = \frac{\boldsymbol{s}_i}{\|\boldsymbol{s}_i\|}$, is

$$\nabla_{\boldsymbol{s}_i} \boldsymbol{\phi}_i = \frac{1}{\|\boldsymbol{s}_i\|} \left(\mathbf{I}_d - \boldsymbol{\phi}_i \boldsymbol{\phi}_i^\top \right).$$

Thus, by the chain rule,

$$\frac{\partial (\boldsymbol{\phi}_i^\top \boldsymbol{v}_t)}{\partial \boldsymbol{s}_i} = \frac{\partial \boldsymbol{\phi}_i}{\partial \boldsymbol{s}_i} \boldsymbol{v}_t = \frac{1}{\|\boldsymbol{s}_i\|} \left(\mathbf{I}_d - \boldsymbol{\phi}_i \boldsymbol{\phi}_i^\top \right) \boldsymbol{v}_t.$$

Therefore, for each i the gradient with respect to $\boldsymbol{s}_i$ is

$$\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{s}_i} = \frac{2\, \boldsymbol{e}_i}{\|\boldsymbol{s}_i\|} \mathbf{P}(\boldsymbol{s}_i)\, \boldsymbol{v}_t = 2\, \boldsymbol{e}_i \, \frac{1}{\|\boldsymbol{s}_i\|} \left(\boldsymbol{v}_t - \frac{\boldsymbol{s}_i \, (\boldsymbol{s}_i^\top \boldsymbol{v}_t)}{\|\boldsymbol{s}_i\|^2} \right), \quad \text{for } i = 1, \ldots, m.$$

Here, the matrix $\mathbf{P}(\boldsymbol{s}_i) = \mathbf{P}(\boldsymbol{\phi}_i) := \left(\mathbf{I} - \frac{\boldsymbol{s}_i \boldsymbol{s}_i^\top}{\|\boldsymbol{s}_i\|^2} \right)$ is known as the *projection matrix onto the orthogonal complement* of $\boldsymbol{s}_i$ in linear algebra (Strang, 2000, §3.3). Stacking these column gradients into the gradient with respect to the matrix $\mathbf{S}$ we obtain

$$\nabla_{\mathbf{S}} \mathcal{L}_t = \left[\frac{\partial \mathcal{L}_t}{\partial \boldsymbol{s}_1}, \, \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{s}_2}, \, \ldots, \, \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{s}_m} \right],$$

Thus, the closed-form gradient can be expressed as:

$$\nabla_{\mathbf{S}} \mathcal{L}_t = \left[\frac{2\left(\boldsymbol{\phi}_1^\top \boldsymbol{v}_t - (\boldsymbol{k}_t)_1 \right)}{\|\boldsymbol{s}_1\|} \left(\mathbf{I}_d - \boldsymbol{\phi}_1 \boldsymbol{\phi}_1^\top \right) \boldsymbol{v}_t, \; \cdots, \; \frac{2\left(\boldsymbol{\phi}_m^\top \boldsymbol{v}_t - (\boldsymbol{k}_t)_m \right)}{\|\boldsymbol{s}_m\|} \left(\mathbf{I}_d - \boldsymbol{\phi}_m \boldsymbol{\phi}_m^\top \right) \boldsymbol{v}_t \right] \tag{25}$$

$$= \left[\frac{2\, \boldsymbol{e}_1}{\|\boldsymbol{s}_1\|} \mathbf{P}(\boldsymbol{s}_1)\, \boldsymbol{v}_t, \; \cdots, \; \frac{2\, \boldsymbol{e}_m}{\|\boldsymbol{s}_m\|} \mathbf{P}(\boldsymbol{s}_m)\, \boldsymbol{v}_t \right] \qquad \square$$

Proposition C.1 (Proof of Proposition 3.1) *Consider the unit sphere $\mathcal{C} = \{\boldsymbol{s} \in \mathbb{R}^d \mid \|\boldsymbol{s}\| = 1\}$ which is a smooth Riemannian manifold. Let $\nabla_s \ell(\mathbf{s})$, the gradient of the loss ℓ, and let $\nabla_{\mathcal{C}} \ell(\mathbf{s})$ be its orthogonal projection from the ambient space $\mathbb{R}^d$ onto the tangent space of the manifold at $\boldsymbol{s}$, denoted by $\mathcal{T}_{\mathcal{C}}(\boldsymbol{s})$.*

In our update rule, the gradient term $\Delta \boldsymbol{s} = \alpha\, \mathbf{h}^{\perp \boldsymbol{s}}$ is orthogonal to $\boldsymbol{s}$; that is, $\Delta \boldsymbol{s} \in \mathcal{T}_{\mathcal{C}}(\boldsymbol{s})$. Hence, we have

$$\nabla_{\mathcal{C}} \ell(\mathbf{s}) = \nabla_s \ell(\mathbf{s})$$

The gradient descent update of ℓ on the Riemannian manifold $\mathcal{C}$ is given by

$$\mathbf{s}_{new} = exp_{\mathbf{s}} \Big(-\eta_t \, \nabla_{\mathcal{C}} \ell(\mathbf{s}) \Big),$$

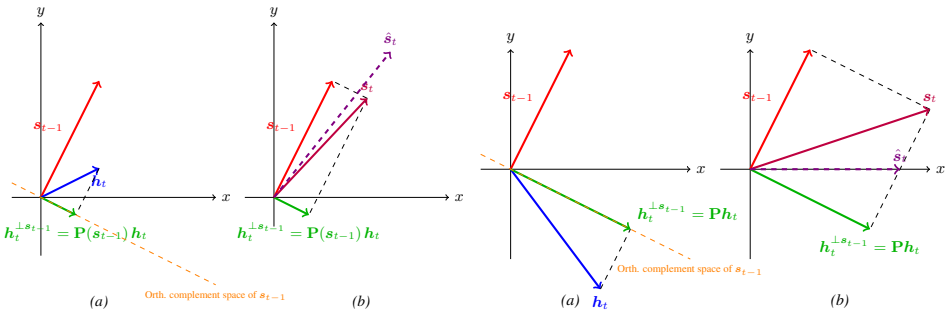

Figure 4: An illustration of the proposed update rule. (a) Example of a single memory slot state, $\boldsymbol{s}_t$, an incoming token representation, $\boldsymbol{h}_t$, and its component orthogonal to the current state, $\boldsymbol{h}_t^{\perp \boldsymbol{s}_{t-1}}$. (b) The updated state according to the proposed update rule, $\boldsymbol{s}_t = \boldsymbol{s}_{t-1} + \alpha_{i,t}\, \boldsymbol{h}_t^{\perp \boldsymbol{s}_{t-1}}$ contrasted with the updated state resulting from the superposition recurrence update used in standard linear attention: $\hat{\boldsymbol{s}}_t = \boldsymbol{s}_{t-1} + \alpha_{i,t}\, \boldsymbol{h}_t$, (dashed arrow). A unit writing intensity ($\alpha_{i,t} = 1$) is assumed for simplicity in both recurrent update rules.

where $\texttt{exp}_{\mathbf{s}}$ *is the exponential map on the sphere* $\mathcal{C}$ *(Absil et al., 2009; Boumal, 2023).*

Replacing the exponential map with its first-order approximation, called retraction step, projects from the tangent space onto the sphere manifold (Bonnabel, 2013). Therefore, our projected gradient update of the form $\boldsymbol{s}_{i,t} = \mathcal{P}_{\mathcal{C}}(\boldsymbol{s}_{i,t-1} + \Delta \boldsymbol{s}_{i,t})$ *(equation 9) is equivalent to performing a Riemannian gradient descent step with retraction on the manifold* $\mathcal{C}$. □

This proposition formalizes that by updating the memory slot with only the orthogonal component and then projecting back onto the unit sphere, we are effectively performing gradient descent on the Riemannian manifold of unit-norm state vectors.

D Experiment Details

Datasets: We trained models on different datasets. For language modeling and common-sense reasoning tasks, models are trained on FineWeb-Edu dataset (Penedo et al., 2024) with context length of 4k. For the sequence length scaling pattern, models are trained on The Pile and Books3 dataset. The Pile is a large-scale, diverse corpus widely used for training and evaluating language models (Gao et al., 2020). It consists of a mixture of high-quality text sources, including books, academic papers, web content, and technical documentation. While it contains relatively few sequences exceeding $8k$ tokens, in this study, we restrict The Pile to a short-context setting with sequence lengths of 2k or 8k tokens. Books3, on the other hand, is a subset of The Pile that consists of high-quality, full-length books, commonly used for training language models for long-context evaluations. In the experiments we used this dataset to test model performance on sequences ranging from 512 to 16k tokens (in increments of $2\times$ per experiment). The same training setup as The Pile is applied to ensure consistency. Since Books3 contains structured narratives and long-form content, it provides a rigorous test of a model's ability to track dependencies over extended contexts.

For all experiments, the training batch size is fixed at 0.5 million tokens, irrespective of sequence length. This means that for a given context length T, each batch contains $0.5M/T$ sequences.

Baseline Models and Model Architecture We compare our method against Transformer++ model (Touvron et al., 2023) as well as the following sub-quadratic sequence models: Linear-Attention (LA) (Katharopoulos et al., 2020), TTT (Sun et al., 2024), DeltaNet (Yang et al., 2024c), Gated DeltaNet (Yang et al., 2024a), Mamba2 (Dao & Gu, 2024). As discussed in the paper, the Lattice layers incorporate ℓ_2-normalization on the state, whereas TTT applies ℓ_2-normalization on the output of the decoding layer.

Model Architecture For sub-quadratic sequence models, we adopt the architectural setup used in Mamba (Gu & Dao, 2023), where each sequence-mixing block consists of a pair of short `Conv1D` layers for the $\{\boldsymbol{q}, \boldsymbol{k}\}$ pair, which share a linear projection. A `GeLU` post-gate is applied to the output

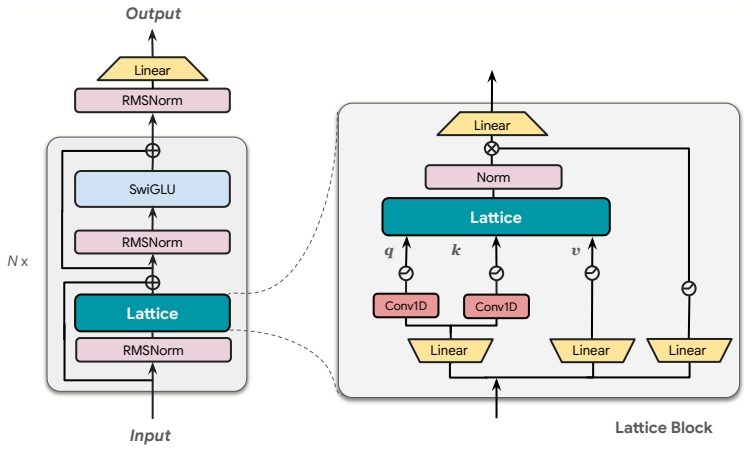

Figure 5: *(Left)* Block diagram of the language model. *(Right)* The Lattice block. Following the architecture used in Mamba (Gu & Dao, 2023), each sequence mixing block is composed of a pair of short `Conv1D` for the pair $\{q, k\}$ and the Lattice is followed by a `GeLU` post-gate.

of the sequence model. The Transformer++ model, on the other hand, follows the architecture proposed in LLaMA (Touvron et al., 2023). All the models follow the multi-head structure introduced in Transformers (Vaswani et al., 2017a). The model architecture used for Lattice is illustrated in Figure 5.

Instead of using a fixed learning rate as in standard gradient descent, we model the learning rate of the compression layer as an input-dependent neural network, $\gamma_t = \text{sigmoid}(\mathbf{W}_\gamma \boldsymbol{x}_t)$. This is a key benefit of the bilevel optimization setup, as the weights of this network can be trained in the outer loop along with the rest of the model weights.

Hyperparameters. The block diagram of the language model is in Figure 5, and the model hyperparameters are listed in Table 4. We use the AdamW optimizer with a cosine learning rate schedule, which includes a warm-up phase of 1k steps (0.5 billion tokens) and a final learning rate of 3e-5. We apply a weight decay of 0.1 and gradient clipping at 1.0. The width of the short convolution layer is set to 4. For language model training on FineWeb-Edu and the reasoning tasks in Table 1, a default chunk size of $C = 4$ is used for Lattice unless otherwise specified. For the ablation studies with small models in Table 2 and the long-context scaling experiments (110M models, Books dataset) a chunk size of $C = 1$ is used to evaluate the exact, non-approximated performance of the compression layer.

Configuration	$n_{\textbf{blocks}}$	$d_{\textbf{model}}$	$d_{\textbf{head}}$	Peak learning rate
110M params / 5B tokens	12	768	64	1e-2
340M params / 15B tokens	24	1024	64	1.5e-3
760M params / 30B tokens	24	1536	64	1.25e-3

Table 4: Model and training hyperparameters.

Performance Scaling Analysis vs Context Length. As illustrated in Table 5, the relative performance gain of Lattice over strong baselines such as Mamba2 and TTT nearly doubles as the context length scales from 512 to 16k tokens. Furthermore, the performance gap widens against other competitive models, including DeltaNet and Gated-DeltaNet (in most ranges), across both the Pile and Books datasets. This trend, alongside consistent results observed in the MQAR task (§D.2), demonstrates that Lattice's Orthogonal State Recurrence (OSR) becomes increasingly important for effective memory compression as sequence length grows.

Table 5: Perplexity (PPL) percentage improvement of Lattice over competitive baselines. Improvement is calculated as $\left(\frac{\text{Baseline PPL}}{\text{Lattice PPL}} - 1\right) \times 100\%$. Lattice shows scaling benefits as context length increases.

Baseline Model	**Pile**		**Books**					
	2k	8k	512	1k	2k	4k	8k	16k
vs. Mamba2	+5.8%	**+8.7%**	+4.6%	+5.6%	+7.0%	+8.1%	+9.7%	**+9.9%**
vs. DeltaNet	+6.8%	+8.5%	+6.4%	+6.8%	+6.9%	+7.1%	**+8.6%**	+7.7%
vs. Gated-DeltaNet	+4.0%	**+5.0%**	+3.7%	+3.9%	+5.0%	+4.5%	**+4.7%**	+4.0%
vs. TTT	+6.5%	**+8.9%**	+5.5%	+6.3%	+7.1%	+7.8%	+8.6%	**+9.8%**

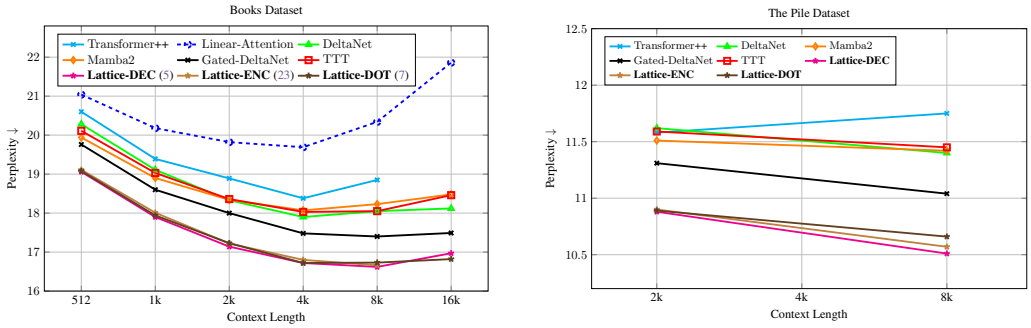

Figure 6: Model perplexity as a function of context length for models of size 110M parameters. *(Left)* displays results for the Books dataset vs context length $\{512, 1024, 2k, 4k, 8k, 16k\}$; *(Right)* shows results for The Pile dataset vs context length $\{2k, 8k\}$. Note that pre-training Transformers from scratch often performs poorly on very long contexts (e.g., 16k); the common approach is finetuning from shorter-context models (Touvron et al., 2023). Therefore, the Transformer results shown here are limited to context lengths $T \leq 8k$.

D.1 Ablations

Scaling Memory Size. Figure 7 illustrates the scaling behavior of Lattice with respect to the number of memory slots m. We observe that the performance of Lattice scales linearly with respect to $\log(m)$ (noting the logarithmic scale on the x-axis). Notably, Lattice with only a quarter of the memory slots ($m = d/4 = 16$) outperforms TTT with full memory ($m = 64$), which empirically validates that the proposed orthogonal update mechanism utilizes the fixed-size state capacity significantly more efficiently, enabling Lattice to achieve $4\times$ compression gain. It is worth noting that while increasing m yields consistent gains, models with $m > d$ come at the cost of increased parameter counts in the embedding projection matrices $\mathbf{W}_q$ and $\mathbf{W}_k$ (and vice versa for models with $m < d$).

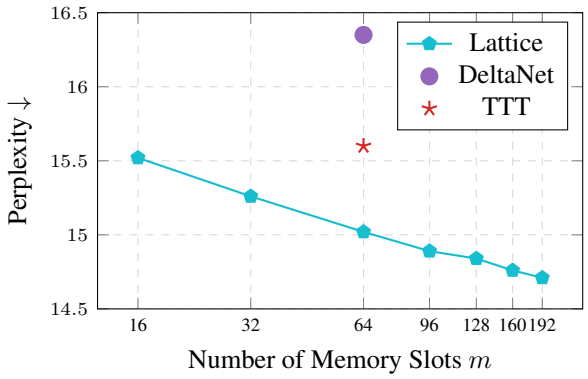

Figure 7: **Ablation on Memory Size (m).** Validation perplexity of Lattice's (110M parameters), trained on FineWeb-Edu with varying number of memory slots m. For comparison, DeltaNet and TTT are plotted at their standard configuration of $m = 64$ (equivalent to head dimension $d = 64$). Lattice consistently outperforms baselines, achieving lower perplexity even with half the memory capacity.

Table 6: Ablation study of Lattice's per-token vs. per-chunk normalization (110M parameters, trained on FineWeb-Edu). The perplexity of the chunk-wise approximation (13) using per-token vs. per-chunk normalization strategies, where the normalization step is only applied after each chunk.

Configuration	ppl ↓
DeltaNet	16.35
TTT C = 1	15.60
Lattice C = 4 Per-token , approx. (13)	15.15
Lattice C = 4 Per-chunk, approx. (13)	15.17
Lattice C = 4 Per-token , approx. (15)	15.12
Lattice C = 4 Per-chunk, approx. (15)	15.13
Lattice C = 16 Per-token , approx. (13)	15.30
Lattice C = 16 Per-chunk, approx. (13)	15.31
Lattice C = 16 Per-token , approx. (15)	15.27
Lattice C = 16 Per-chunk, approx. (15)	15.38

Per-token vs. Per-chunk Normalization. Table 6 compares the perplexity of the chunk-wise approximations (13) using per-token versus per-chunk normalization strategies. In the per-chunk variant, normalization is applied solely at chunk boundaries while the orthogonalization computations are still performed per-token. For the first-order approximation equation 15, the results indicate that the performance divergence between the two strategies widens as the chunk size increases. These results highlight the importance of per-token normalization for ensuring both performance and stability across larger chunk sizes. We attribute this to the fact that applying normalization exclusively at chunk boundaries creates a geometric discrepancy between the intra-chunk and inter-chunk update dynamics. On the other hand, applying per-chunk normalization to the zeroth-order approximation (13) reduces the intra-chunk update to a simpler recurrence with a scalar-valued decay which shows consistent results across both chunk sizes. As demonstrated by the throughput analysis in Figure 9, this per-chunk normalization variant successfully optimizes the efficiency-expressivity trade-off, offering competitive training speeds that rival baselines like Mamba2 and TTT.

D.2 Recall-Intensive Task: Multi-Query Associative Recall

To rigorously evaluate the memory capacity and in-context learning capabilities of Lattice, we evaluate the model on Multi-Query Associative Recall (MQAR) task introduced by Arora et al. (2023).

Prior research indicates that a model's capability in associative recall task is strongly predictive of its in-context learning quality and language modeling performance (Olsson et al., 2022). While standard associative recall tests a model's ability to retrieve a value for a single query after a sequence of key-value pairs, MQAR introduced by Arora et al. (2023) as a more challenging and realistic scenario where the model must perform multiple recalls at varying positions within a single forward pass. In this task, a sequence consists of a series of key-value pairs followed by a series of queries (keys) mixed with other tokens. The model must correctly predict the value associated with a specific key. An example sequence is formatted as follows:

$$\text{A 4 B 3 C 6 F 1 E 2} \ldots \text{A ? C ? F ? E ? B ?} \rightarrow \text{4 6 1 2 3}$$

In this setup, the model must compress the association of the key-value pairs in its memory state over potentially long distances and retrieve them correctly when queried multiple times. Therefore, MQAR serves as a strong benchmark for the memory capacity of recurrent architectures while Transformer architecture can easily solve this task.

Experimental Setup. We follow Arora et al. (2023) and adopt the experimental setup used in Beck et al. (2024) for the more difficult setting. We generate 100,000 synthetic training samples and 3,000 validation samples from a vocabulary size of 8192. All models are trained for 64 epochs using the

AdamW optimizer with a cosine annealing learning rate schedule after a 10% linear warm-up phase and with a weight decay of 0.1.

We train models with two blocks using a single head, while varying the Model Dimension (d) to observe scaling behaviors. We set the number of slots $m = d$ for Lattice, TTT, DeltaNet, and GLA; consequently, each block maintains a state matrix of size d^2. In comparison, xLSTM uses an additional normalizer state vector, resulting in a larger state size of $d^2 + d$ per block.

We evaluate Lattice on two challenging settings across different context lengths (T) and high numbers of key-value pairs. For each setting, we sweep over learning rate grids to ensure optimal convergence:

- Context Length T=1024: We sweep learning rates over {1e-2, 3.16e-3, 1e-3, 3.16e-4, 1e-4} with a batch size=96.
- Context Length T=2048: We sweep learning rates over {1e-2, 1e-3, 2.2e-4, 5e-5, 1e-5} with a batch size=24.

For each context length, we scale the difficulty by varying the number of Key-Value pairs that must be memorized, testing $N_{KV} \in \{48, 96, 256\}$. This progression specifically evaluates memory capacity of the model, determining whether the architecture can effectively store and recall a high density of information within the fixed-size state, thereby testing the efficiency of the compression mechanism proposed by Lattice.

As the results in Figure 8 show, Lattice offers the best MQAR accuracy over the majority of the settings and it exhibits a significant improvement over its main counterparts: TTT and DeltaNet. Furthermore, while baselines exhibit performance degradation as context length increases, Lattice maintains its accuracy across both context lengths. *This overall shows the efficient compression mechanism of Lattice, demonstrating its superior capability to store and retrieve dense information within its fixed-size recurrent memory.*

D.3 Throughput and Scalability Analysis

The training throughput are compared in Figure 9. To isolate the algorithmic efficiency of each recurrence mechanism, all recurrent models are implemented in pure JAX without relying on specialized hardware-optimized kernels, whereas the Transformer++ baseline leverages highly optimized implementation of Flash-Attention-2 kernels (Dao, 2023) in Pallas with a block size of 256. While these models can be further accelerated with hardware-specific kernels (e.g., Triton or Pallas), our focus is to demonstrate the inherent scalability of the Lattice mathematical formulation. The results demonstrate that Lattice exhibits superior scalability compared to Transformers as sequence length increases, while maintaining a scaling trajectory comparable to baselines like TTT and Mamba2. Specifically, the per-chunk normalization variant (Lattice per-chunk) rivals these efficient baselines through further optimizing the efficiency-expressivity trade-off by applying normalization per-chunk while the orthogonalization computations are still performed per-token. In contrast, Gated-DeltaNet suffers from the computational overhead of solving lower triangular systems during its recurrence—a bottleneck that becomes increasingly restrictive as chunk size grows. Collectively, these findings confirm that Lattice maintains the favorable sub-quadratic scaling characteristic of RNNs while offering improved expressivity through its unique state-dependent orthogonal updates.

Future Works. The proposed chunk-wise approximations in Equations (13) and (15) enable parallelizable intra-chunk computation using standard, hardware-efficient operators such as matrix and element-wise multiplications. However, the non-linear nature of the test-time training in general prevents inter-chunk parallelization via associative scans. We believe that exploring more hardware-efficient strategies remains a key area for future work. Specifically, adapting techniques such as the large-chunk training strategies proposed in LaCT (Zhang et al., 2025), or the multi-stage hierarchical process (Li et al., 2025) to Lattice represents a promising direction to maximize hardware utilization while retaining the benefits of the orthogonal update rule. In this work, *we focused primarily on introducing a novel, interpretable recurrent mechanism for efficient memory compression, leaving further optimization of training throughput to future research*. Additionally, adapting Lattice for efficiently fine-tuning pre-trained Transformers into memory-efficient RNNs presents a compelling direction for future work.

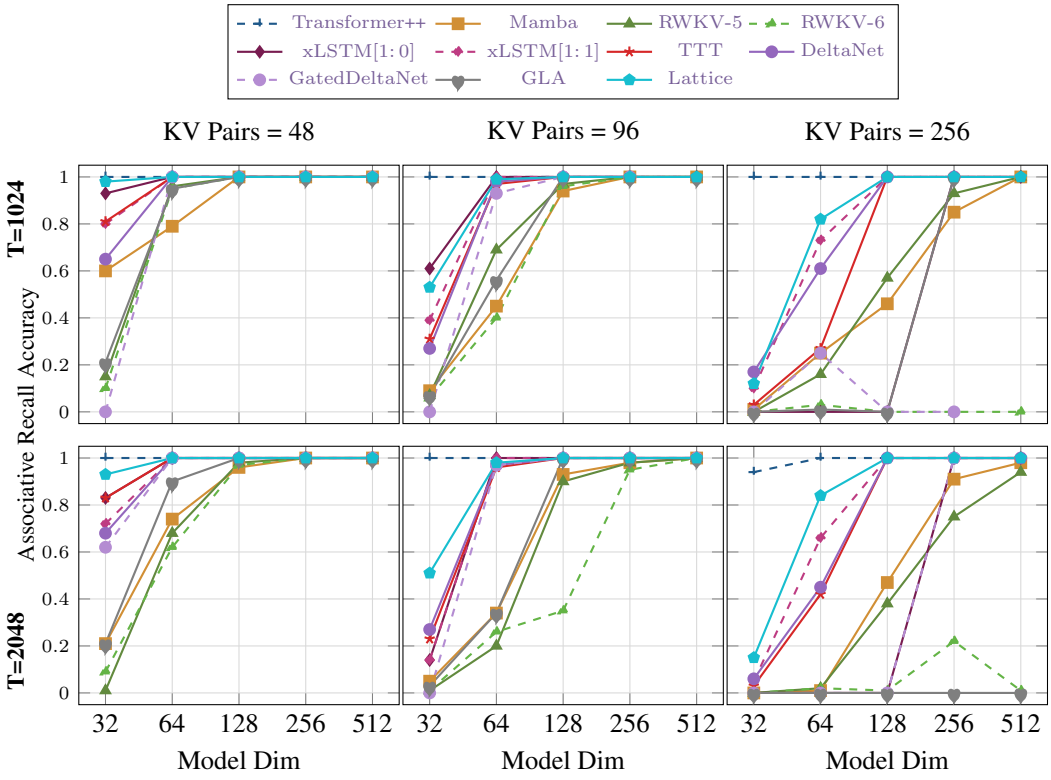

Figure 8: Associative Recall Accuracy (%) on the recall-intensive MQAR task. The baseline results for Transformer, Mamba (Gu & Dao, 2023), RWKV-5/6 (Peng et al., 2023; 2024), xLSTM[1:0], and xLSTM[1:1] are taken from (Beck et al., 2024), while we trained Lattice, TTT, DeltaNet, GatedDeltaNet, and GLA. Notably, GatedDeltaNet and GLA were unable to effectively learn this task for the 256 Key-Value pair setting across most model dimensions.

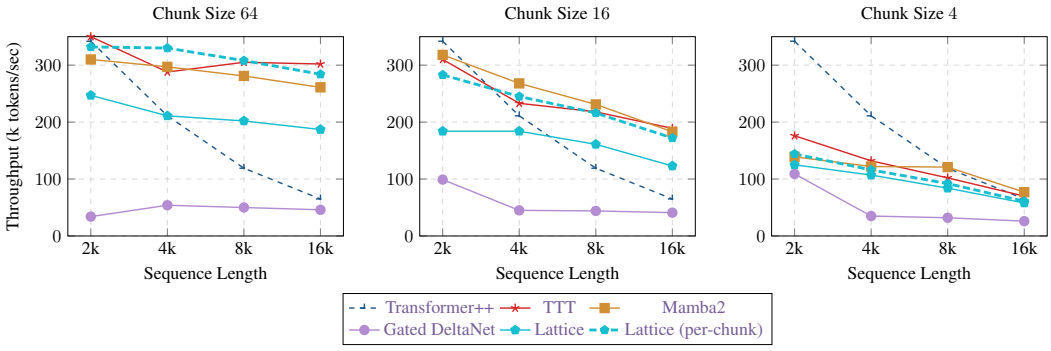

Figure 9: **Training Throughput vs. Sequence Length.** Throughput (in k tokens/sec) for 110M parameter models trained on a 2×2 TPU Trillium setup. Transformer++ uses optimized Flash-Attention2 with a block size of 256. Due to its recurrent formulation, Lattice maintains higher throughput for $C \in \{16, 64\}$ than Transformer++ as sequence length increases, remaining competitive with TTT and Mamba2. Lower chunk sizes generally reduce throughput due to reduced parallelism.

E IMPLEMENTATION

```

from functools import partial
import jax
import jax.numpy as jnp
from flax import linen as nn
from jax import lax, vmap
from einops import rearrange
from jax.numpy.linalg import norm

@partial(jax.jit, static_argnames=['mode', 'chunk_size'])
def lattice_compress(S0, K, V, Q, gamma, mu, mode='Dec', chunk_size=16):
    """
    A simple Lattice operation with recursive update.
    S0: (BxHxMxD)
    K: (BxLxHxM), V: (BxLxHxD), Q: (BxLxHxM), gamma/mu: (BxLxHx1)
    """
    # split chunks and transpose to B H NC C d
    K = rearrange(K, 'b (nc c) h m -> b h nc c m', c=chunk_size)
    V = rearrange(V, 'b (nc c) h d -> b h nc c d', c=chunk_size)
    Q = rearrange(Q, 'b (nc c) h m -> b h nc c m', c=chunk_size)
    gamma = rearrange(gamma, 'b (nc c) h 1 -> b h nc c 1', c=chunk_size)
    mu = rearrange(mu, 'b (nc c) h 1 -> b h nc c 1', c=chunk_size)

    if chunk_size == 1:  # scan over sequence
      scan_fn = partial(compress_step, mode=mode)
    else:  # scan over chunks
      scan_fn = partial(compress_chunk, mode=mode)

    def compress_scan(s0, k,v,q,e,m):
        return lax.scan(scan_fn, s0, (k,v,q,e,m))

    # vmap over batch and heads
    batch_head_scan_fn = vmap(
        vmap(compress_scan, axis_name="heads"),
        axis_name="batch")
    S, Y = batch_head_scan_fn(S0, K, V, Q, gamma, mu)
    Y = rearrange(Y, 'b h nc c d -> b (nc c) h d')
    return S, Y

def compress_step(S, inputs_t, mode='Dec'):
  # S: MxD
  # inputs_t: k:1xM, v:1xD, q:1xM, gamma:1x1, mu:1x1
  k, v, q, gamma, mu = inputs_t

  s_norm = norm(S, axis=-1, keepdims=True) + 1e-6
  Phi = S / s_norm

  if mode == 'Dec':
    h = k @ Phi - v
  elif mode == 'Sim':
    h = - v

  # 1) orthogonal projection  h_t^{⊥s_i}  (7), (8) : compute Delta   Equation 12
  h_hat= (h @ Phi.T) / s_norm.T
  k_hat = k / s_norm.T
  Delta1 = k_hat.T @ h
  Delta2 = S * (h_hat * k_hat).T
  Delta = Delta1 - Delta2

  # #VERIFY ORTHOGONALITY of Delta and S
  # jax.debug.print("<Delta,S>: {}",jnp.einsum('md, md->m',Delta,S))

  # 2) update S and normalize it  β_t
  S_t = mu * S - gamma * Delta
  beta = s_norm * jax.lax.rsqrt(
      ((mu*S)**2 + (gamma * Delta)**2).sum(axis=-1, keepdims=True))
  # beta = s_norm / (norm(S_t, axis=-1, keepdims=True)+ 1e-6)
  S_t *= beta       #  (Equation 9)
  # readout
  y = q @ S_t
  return S_t, y

def compress_chunk(S, inputs_t, mode='Dec'):
  # S: MxD (the code operates on the transpose of the text's state matrix S ∈ R^{d×m}
  # inputs_t: K:CxM, V:CxD, Q:CxM, gamma:Cx1, mu:Cx1
  K, V, Q, gamma, mu = inputs_t

  s_norm = norm(S, axis=-1, keepdims=True) + 1e-6 # Mx1
```

```
Phi = S / s_norm # MxD

if mode == 'Dec':
  H = K @ Phi - V  # CxD
elif mode == 'Sim':
  H = - V  # CxD
H_hat= (H @ Phi.T) / s_norm.T  # CxM :  Equation 12
K_hat = K / s_norm.T  # CxM

# compute  ||ΔS||^2 and β Equation 21
Delta1_sq = K_hat**2 * (H*H).sum(axis=-1)[:, None] # CxM
Delta2_sq = (s_norm.T**2) * (H_hat * K_hat)**2     # CxM
Delta_sq = jnp.clip(Delta1_sq - Delta2_sq, min=0.) # CxM
Delta_sq = gamma**2 * Delta_sq
beta = s_norm.T * jax.lax.rsqrt(mu**2 * s_norm.T**2 + Delta_sq)   # (CxM)

G_gate = beta * (mu + gamma*(H_hat*K_hat)) # CxM :  Equation 15

# Parallel intra-chunk based on linear attention
# with vectorized forgetting gate (GLA):  Equation 15
K_tilde = -beta * gamma * K_hat  # CxM
S_t, Y = GLA_intra_chunk(Q=Q, K=K_tilde, V=H, G=G_gate, S0=S)
return S_t, Y
```

Listing 1: A simple implementation of the Lattice update rule.

LLM Usage. We acknowledge the use of a large language model (LLM) solely for improving the linguistic quality and clarity of this manuscript and polishing the paper.