LATTICE: Learning to Efficiently Compress the Memory

Anonymous Authors[1]

Abstract

Attention mechanisms have revolutionized sequence learning but suffer from quadratic computational complexity. This paper introduces Lattice, a novel recurrent neural network (RNN) mechanism that leverages the inherent low-rank structure of K-V matrices to efficiently compress the cache into a fixed number of memory slots, achieving sub-quadratic complexity. We formulate this compression as an online optimization problem and 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 memory update process. The core innovation is the orthogonal update: each memory slot is updated exclusively with information orthogonal to its current state hence incorporation of only novel, non-redundant data, which minimizes the interference with previously stored information. The experimental results show that Lattice achieves the best perplexity compared to all baselines across diverse context lengths, with performance improvement becoming more pronounced as the context length increases.

1. Introduction

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 alternative to softmax attentions. While traditional SSMs, with their inherent linear recurrent structure, offer parallelization during training, they often struggle to match the expressivity of standard attention. Linear attention methods reduce complexity by approximating the attention matrix but can sacrifice accuracy. More recently, input-dependent SSMs (Gu & Dao, 2023; Dao & Gu, 2024) and modern gated RNNs (Orvieto et al., 2023; De et al., 2024; Beck et al., 2024) have demonstrated enhanced expressiveness and improved in-context learning while enabling parallelization through techniques like associative scan (Blelloch, 1990; Smith et al., 2023; De et al., 2024). However, a fundamental challenge remains: their ability to efficiently compress and summarize information over very long contexts is often limited by their fixed-size hidden states (Arora et al., 2024). Moreover, their linear updates to memory lack efficient mechanisms for selective interaction between stored information and incoming keys, limiting their ability to discard irrelevant or redundant content dynamically. On the other hand, 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 by dynamically adapting convolutional filters to the input, but they are not inherently compatible with causal modeling, used in autoregressive language generation.

The update rule in the (gated) linear attention, and its gated variant, typically relies on an additive outer product of input-dependent representations, which can be generally expressed as: $\mathbf{S}_t = \mathbf{S}_{t-1} + f_g(\boldsymbol{x}_t) \otimes f_v(\boldsymbol{x}_t)$ 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.[1] While this *linear rank-one modification* to the state matrix (also referred to as Hebbian-like update rule) enables efficient parallel computation, it suffers from a key limitation: the additive update term in the recurrence is not directly aware of the current memory state $\mathbf{S}_{t-1}$ and operates independently of it. This lack of state awareness can cause *key interference* and eventually lead to *overcapacity regime* (Schlag et al., 2021), where multiple tokens attempt to write to the same memory slot when the size of memory is shorter than the sequence length.

Based on this insight, ideally, the writing intensity of the t-th token $\boldsymbol{x}_t$ to the j-th memory slot, $(\mathbf{S})_{j,:}$, 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

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

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

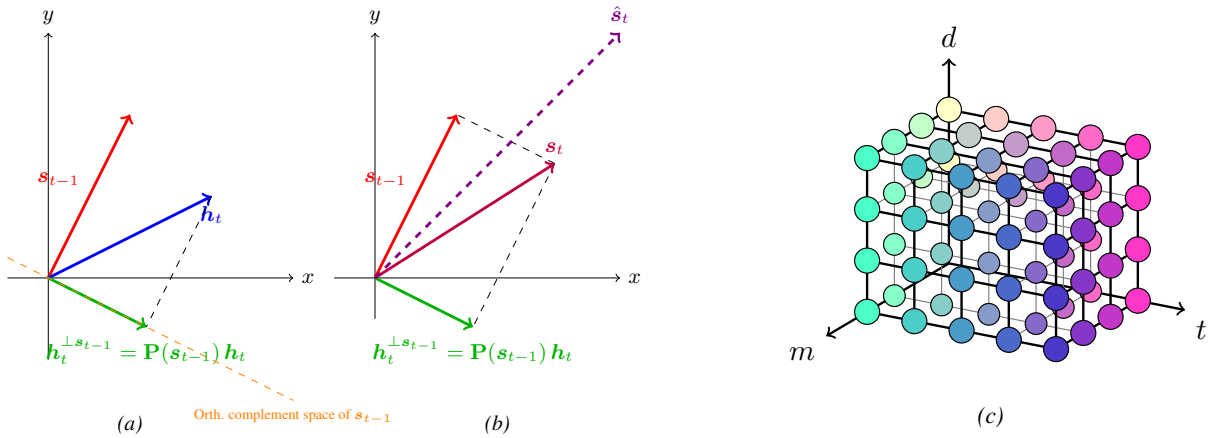

Figure 1: A geometric visualizing of the proposed update rule. *(a)* A single current state vector, $\boldsymbol{s}_{t-1} = \mathbf{S}_{t-1}[:, i]$, an incoming token representation, $\boldsymbol{h}_t$, and its component orthogonal to the current state, $\boldsymbol{h}_t^{\perp \boldsymbol{s}_{t-1}}$. *(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.

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. In the following section, we approach this problem by framing it as an online optimization problem and drive an optimal update rule to compress and retain essential information from a sequence.

2. Compression Layer

State-Dependent Compression for Unbounded Caches Our objective is to develop a compression model that dynamically updates and maintains a compact representation of the contextual history—encoded 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 computationally efficient querying, as the memory read-out is processed using the compressed state, i.e., $\boldsymbol{y}_t = \mathbf{S}_t \hat{\boldsymbol{r}}_t$ instead of querying from the full cache. Here, $\hat{\boldsymbol{r}}_t$ represents a retrieval vector analogous to the attention weights in the standard attention layer.

This lossy compression approach involves a trade-off between computational efficiency, memory usage, and query precision. A more compact memory representation (i.e., smaller m) reduces 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 the precision loss. We can formulate this problem as reconstructing the input as: $\mathbf{x}_t \approx \tilde{\mathbf{x}}_t = \mathbf{S}_t \boldsymbol{k}_t$, where $\tilde{\mathbf{x}}_t$ is the reconstructed input, $\mathbf{S}_t$ represents the dynamically updated memory matrix, and $\boldsymbol{k}_t$ is a latent representation vector.

Inspired by classical representation learning techniques such as dictionary learning, sparse coding, and structured matrix factorization (Mairal et al., 2009; Lyu et al., 2020)[2], 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).

2.1. Decoding Layer

For each input sequence, we model a *decoding layer*, denoted as $g(\boldsymbol{k}_t; \mathbf{S}_t)$, operating 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. This makes it 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 $\boldsymbol{v}_t$. Since, the goal is to minimize the reconstruction error, we formulate an optimization problem that minimizes the following ℓ_2 loss as its objective at each time step:

$$\mathcal{L}_t = \|g(\boldsymbol{k}_t; \mathbf{S}_t) - \boldsymbol{v}_t\|^2, \quad \mathbf{S}_t \in \mathbb{R}^{d \times m}, \ \boldsymbol{v}_t \in \mathbb{R}^d, \ \boldsymbol{k}_t \in \mathbb{R}^m$$

This is referred to as *compression loss* throughout this paper. The latent representation $\boldsymbol{k}_t$ is generated by a model-based

[2]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).

encoder network, implemented as a linear projection of the input: $\boldsymbol{k}_t = \mathbf{W}_k \mathbf{x}_t$ where $\mathbf{W}_k \in \mathbb{R}^{m \times d_x}$ is a projection weight matrix. This weight remains fixed during the internal state updates and is trained jointly with the rest of model parameters in the outer training loop. This setup aligns with meta-learning frameworks (Schmidhuber, 1992; Thrun & Pratt, 1998; Andrychowicz et al., 2016) 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) outer model parameters, including the projection layer weights, collectively denoted as $\boldsymbol{\mathcal{W}}$, which capture the broader patterns in the training set. This leads to a bilevel learning process, composed of:

- *Inner Loop (State Update):* A fast update level that adapts the internal states $\mathbf{S}_t$ for each token within a sequence by minimizing the compression loss in equation **??**. Each sequence effectively serves as a 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 regular 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 level loop typically employs standard optimizers such as ADAM (Kingma & Ba, 2014) and learns generalizable patterns in the training dataset.

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

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

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 tokens, ensuring a causal and context-dependent evolution of the internal state.

The gradient of the reconstruction loss with respect to $\mathbf{S}_{t-1}$ can be computed using the chain rule:

$$\begin{aligned}\nabla_{\mathbf{S}} \mathcal{L}_t &= \mathbf{G}_t(\mathbf{S}_{t-1}, \boldsymbol{k}_t, \boldsymbol{v}_t) \\ &= 2\left(g(\boldsymbol{k}_t; \mathbf{S}_{t-1}) - \boldsymbol{v}_t\right)^\top \nabla_S g(\boldsymbol{k}_t; \mathbf{S}_{t-1})\end{aligned} \tag{2}$$

where: $\mathbf{J}_g := \nabla_S g(\boldsymbol{k}_t; \mathbf{S}_{t-1})$ is the Jacobian of $g(\cdot)$. In practice, the term $\left(g(\boldsymbol{k}_t; \mathbf{S}_{t-1}) - \boldsymbol{v}_t\right)^\top \mathbf{J}_g$ can be efficiently computed using the vector-Jacobian product (`vjp`) functionality available in modern machine learning frameworks. This avoids explicitly forming the full Jacobian matrix and leverages efficient automatic backpropagation.

2.1.1. STATE NORMALIZATION

In dictionary learning and subspace learning, normalizing basis vectors (dictionary atoms or principal components) is a common practice. Motivated by this, we apply column-wise normalization to each state vector within the state matrix. Consequently, the decoding function is defined as:

$$\hat{\boldsymbol{v}}_t = g(\boldsymbol{k}_t; \mathbf{S}_t) = \phi(\mathbf{S}_t)\boldsymbol{k}_t$$

with the corresponding reconstruction loss:

$$\mathcal{L}_t = \|\phi(\mathbf{S}_t)\boldsymbol{k}_t - \boldsymbol{v}_t\|^2, \;\; \boldsymbol{v}_t \in \mathbb{R}^d, \; \boldsymbol{k}_t \in \mathbb{R}^m \tag{3}$$

This implies that at each time step, the internal states of the compression layer are updated to ensure that the linear combination of the normalized state vectors closely approximates the target vector $\boldsymbol{v}_t$. To derive the closed-form for gradient of this objective, we define: $\Phi = [\boldsymbol{\phi}_1, ..., \boldsymbol{\phi}_m]$, where $\boldsymbol{\phi}_i = \frac{\boldsymbol{s}_i}{\|\boldsymbol{s}_i\|}$ and $\boldsymbol{s}_i$ is i-th column of the state (i.e., the i-th basis vector), and define the reconstruction error as $\boldsymbol{e}_t := \phi(\mathbf{S}_{t-1})\boldsymbol{k}_t - \boldsymbol{v}_t$, that is the difference between the decoding based on the current state and the target, $\boldsymbol{v}_t$. Then, by the chain rule, the gradient of the loss with respect to $\mathbf{S}$ is given by:

$$\begin{aligned}\nabla_{\mathbf{S}} \mathcal{L}_t &= \boldsymbol{e}_t^\top \left[k_{t_1} \mathbf{J}_\phi(\boldsymbol{s}_1), \quad \ldots, \quad k_{t_m} \mathbf{J}_\phi(\boldsymbol{s}_m)\right] \\ &= \boldsymbol{e}_t^\top \times_1 \left[\mathbf{J}_\phi(\boldsymbol{s}_1), \quad \ldots, \quad \mathbf{J}_\phi(\boldsymbol{s}_m)\right] \odot \boldsymbol{k}_t^\top \end{aligned} \tag{4}$$

$$\text{where } \mathbf{J}_\phi(\boldsymbol{s}_i) = \frac{\mathbf{P}(\boldsymbol{s}_i)}{\|\boldsymbol{s}_i\|} = \frac{1}{\|\boldsymbol{s}_i\|}\left(\mathbf{I} - \frac{\boldsymbol{s}_i \boldsymbol{s}_i^\top}{\|\boldsymbol{s}_i\|^2}\right). \tag{5}$$

where $\mathcal{G} := \left[\mathbf{J}_1, \ldots, \mathbf{J}_m\right]$ is a $d \times d \times m$ tensor formed by stacking $\mathbf{J}_j$ along the last dimension. The vector-tensor product is defined as $\mathbf{Q} = \boldsymbol{e}^\top \times_1 \mathcal{G} = \left[\boldsymbol{e}^\top \mathbf{J}_1, \ldots, \boldsymbol{e}^\top \mathbf{J}_m\right]$[3]. In this equation, 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).

This derivation reveals an *interesting and interpretable update rule*. A key insight is that each slot in memory (column of the state $\boldsymbol{s}_i$) is nonlinearly updated, by the the projection of the reconstruction error, $\boldsymbol{e}_t$ onto the space orthogonal to the that slot $\boldsymbol{s}_i$. The update rule suggests an interpretable decomposition of the $\boldsymbol{e}_t$ into two components: I) $\boldsymbol{e}_t^{\perp \boldsymbol{s}_i}$, **The component orthogonal to $\boldsymbol{s}_i$**, which is used to update the memory slot. II) $\boldsymbol{e}_t^{\| \boldsymbol{s}_i}$, The component of $\boldsymbol{e}_i$ aligned with $\boldsymbol{s}_i$, which is discarded in the update rule, ensuring non-redundant updates.

[3] This operation is implemented using `torch.einsum('i,ikj->kj', e, G)` in `PyTorch`.

This implies that *each memory slot 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 contribution of the t-th token to the i-th memory slot. Figure 1 visualizes this orthogonal update rule.

In the following, we explore two related formulation of the compression problem.

2.2. 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:

$$\mathcal{L}_t = \left\| \phi(\mathbf{S}_t)^\top \boldsymbol{v}_t - \boldsymbol{k}_t \right\|^2, \quad \boldsymbol{v}_t \in \mathbb{R}^d, \ \boldsymbol{k}_t \in \mathbb{R}^m \tag{6}$$

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

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \gamma_t \boldsymbol{v}_t^\top \times_1 \left[\frac{\mathbf{P}(\boldsymbol{s}_1)}{\|\mathbf{s}_1\|}, \ldots, \frac{\mathbf{P}(\boldsymbol{s}_m)}{\|\mathbf{s}_m\|} \right] \odot \boldsymbol{e}_t^\top \tag{7}$$

An alternative approach to encoding is to maximize the dot-product similarity between the encoded representation and the target vector, leading to the following objective:

$$\mathcal{L}_t = -\langle \phi(\mathbf{S}_t)^\top \boldsymbol{v}_t, \ \boldsymbol{k}_t \rangle, \quad \boldsymbol{v}_t \in \mathbb{R}^d, \ \boldsymbol{k}_t \in \mathbb{R}^m \tag{8}$$

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \gamma_t \boldsymbol{v}_t^\top \times_1 \left[\frac{\mathbf{P}(\boldsymbol{s}_1)}{\|\mathbf{s}_1\|}, \ldots, \frac{\mathbf{P}(\boldsymbol{s}_m)}{\|\mathbf{s}_m\|} \right] \odot \boldsymbol{k}_t^\top \tag{9}$$

General form. The formulations presented in Eqs. (4, 7, and 9) offer principled approaches for designing compression layers in our framework. In general, we refer to this approach as *Orthogonal State Recurrence (OSR)*, which unifies these update rules into a common framework formulated as follows

$$\begin{aligned} \{\boldsymbol{y}_t\}_{t=1}^T &= \mathrm{OSR}(\{\boldsymbol{k}_t, \boldsymbol{v}_t, \boldsymbol{q}_t\}_{t=1}^T) \\ &:= \begin{cases} \mathbf{S}_t = \mathbf{S}_{t-1} - \gamma_t \boldsymbol{h}_t^\top \times_1 \left[\frac{\mathbf{P}(\boldsymbol{s}_1)}{\|\mathbf{s}_1\|}, \ldots, \frac{\mathbf{P}(\boldsymbol{s}_m)}{\|\mathbf{s}_m\|} \right] \odot \boldsymbol{c}_t^\top \\ \boldsymbol{y}_t = \mathbf{S}_t \boldsymbol{q}_t \end{cases} \end{aligned} \tag{10}$$

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.

2.3. Stabilizing Memory Updates via Normalization

At each recurrence step, each 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 $\Delta \boldsymbol{s}_{i,t} := \alpha_{i,t} \, \boldsymbol{h}_t^{\perp \boldsymbol{s}_{i,t-1}}$, with $\boldsymbol{h}_t^{\perp \boldsymbol{s}_{i,t-1}}$ denoting the component of the incoming token that is orthogonal to $\boldsymbol{s}_{i,t-1}$ and $\alpha_{i,t}$ an input-dependent writing intensity. While this update scheme avoids interfering with the existing memory, through *adding only novel, non-redundant information*, it leads to a monotonic increase in the norm of $\boldsymbol{s}_i$ with each update, as shown by the Pythagorean theorem: $\|\boldsymbol{s}_{i,t}\|^2 = \|\boldsymbol{s}_{i,t-1}\|^2 + \|\Delta \boldsymbol{s}_{i,t}\|^2$. This can cause 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 a unit sphere $\mathcal{C} = \{\boldsymbol{s} \in \mathbb{R}^d \mid \|\boldsymbol{s}\| = 1\}$, and enforce this constraint by projecting the Euclidean update back onto $\mathcal{C}$, denoted by $\mathcal{P}_{\mathcal{C}}(\cdot)$, at each time step. Therefore, the effective update becomes

$$\boldsymbol{s}_{i,t} = \mathcal{P}_{\mathcal{C}}(\boldsymbol{s}_{i,t-1} + \Delta \boldsymbol{s}_{i,t}) = \beta_{i,t} \left(\boldsymbol{s}_{i,t-1} + \Delta \boldsymbol{s}_{i,t} \right), \tag{11}$$

where $\beta_{i,t} = \left(1 + \|\Delta \mathbf{s}_{i,t}\|^2\right)^{-\frac{1}{2}}$, assuming $\|\mathbf{s}_{i,t-1}\| = 1$. This normalization step, achieved by multiplying with the scalar $\beta_{i,t}$, ensures that the updated state $\boldsymbol{s}_{i,t}$ remains within a bounded region while preserving the steepest-descent direction, thus maintaining stability and allowing the model to effectively store relevant information. In addition, this normalization of the recurrence terms acts analogously to a forgetting gate in RNNs, and also normalizes the step size of the update term, a technique known to improve convergence in optimization algorithms such as Adagrad (Duchi et al., 2011) and Adam (Kingma & Ba, 2014). In the experiments, we initialize the state matrix, $\mathbf{S}_0$, with orthonormal columns. In the following proposition, we formalize the relation between the proposed *Normalized Orthogonal State Recurrence* (NOSR) and Riemannian optimization (Absil et al., 2009; Boumal, 2023).

Proposition 2.1 (Equivalence to Gradient Descent on Riemannian Manifold)**.** *Let $\mathcal{C} = \{\mathbf{s} \in \mathbb{R}^d \mid \|\mathbf{s}\| = 1\}$ be 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 11), where the update term $\Delta \boldsymbol{s}_{i,t}$ lies in the subspace orthogonal to $\boldsymbol{s}_{i,t-1}$ (cf. equation 10), is equivalent to a retraction step in Riemannian optimization (Bonnabel, 2013).*

3. Experiments

We evaluate the proposed architecture across multiple language modeling tasks, benchmarking its performance on both short-context and long-context datasets. Experiment details are provided in Appendix D.

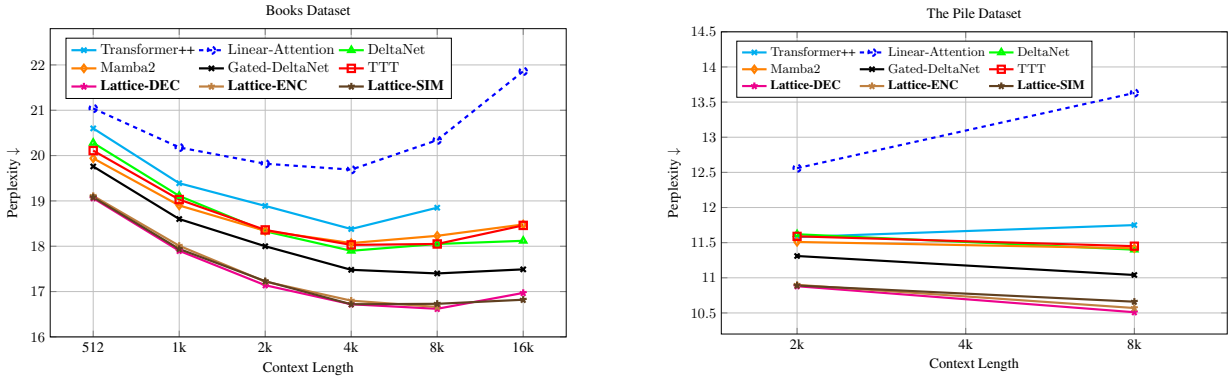

Figure 2: Model perplexity as a function of context length for models of size 110M parameters. *(Left)* : results for the Books dataset vs context length $\{512, 1024, 2k, 4k, 8k, 16k\}$; *(Right)* : results for The Pile dataset vs context length $\{2k, 8k\}$. Note that pre-training Transformers from 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 baseline pre-trained Transformer results shown here are limited to context lengths $T \leq 8k$.

Table 1: Performance comparison of language models of size 340M parameters trained on the Pile dataset (context length 2k, 7.5B tokens). Results include language modeling perplexity on the test set (first column) and accuracy of the trained models on zero-shot common-sense reasoning tasks.

Model	Pile ppl ↓	LMB. acc ↑	PIQA acc ↑	Hella. acc ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc ↑	Avg. ↑
340M params / 7.5B tokens								
Transformer++	8.48	30.64	52.01	26.47	49.57	28.75	22.4	34.97
Linear-Attention	9.04	29.33	55.86	27.33	49.64	34.33	23.78	36.71
Mamba2	8.62	30.78	57.07	28.83	51.54	35.94	22.92	37.85
DeltaNet	8.67	30.71	58	28.93	49.33	35.52	24.03	37.75
Gated-DeltaNet	8.58	30.07	57.51	28.77	50.91	35.73	22.83	37.64
TTT	8.65	30.44	56.91	28.5	50.99	34.25	23.09	37.36
Lattice-DEC (4)	8.27	31.33	57.94	29.67	51.07	36.53	24.29	38.47
Lattice-ENC (7)	8.28	31.18	57.23	29.5	50.51	35.77	22.49	37.78
Lattice-SIM (9)	8.28	31.2	57.51	30.07	50.99	35.94	23.86	38.26

Table 2: Ablation study evaluating improvements upon the linear DeltaNet architecture (Yang et al., 2024b). All models have 125M parameters and were trained on The Pile dataset. The last row corresponds to the final Lattice configuration.

Configuration	ppl ↓
DeltaNet (Yang et al., 2024b)	11.62
TTT (Sun et al., 2024)	11.59
Lattice	
+ orthogonal recurrence (4)	11.26
+ normalized projection (11)	10.94
+ forget-gate (15)	10.94

The results reported in Table 4 and Table 1 (also in Figure 2) demonstrate that Lattices consistently achieve the best perplexity compared to all baselines across a range of context lengths. Importantly, the performance gains of Lattice relative to other linear RNNs become more pronounced as the sequence length grows. This trend highlights the promise of the proposed approach for tasks requiring long-context, where the proposed effective memory management is crucial for maintaining model expressivity and efficiency. Furthermore, the performance of the trained models on various zero-shot common sense reasoning tasks are reported in table Table 1. As the results show, Lattice outperforms all the baseline models on these benchmarks, achieving the highest average accuracy.

Ablation. In this study, we ablate key components of the Lattice to evaluate the contribution of each to the overall performance. The results in Table 2 underscore the significance of state normalization (Equation 9) and normalized projection (Equation 11) to the model's overall performance. Furthermore, our analysis indicates that the forget gate's impact on the overall performance is negligible, suggesting that the normalized projection introduced in Equation 11 inherently acts as a forgetting mechanism.

4. Conclusion

This work introduced a novel recurrent neural network mechanism designed for efficient information compression into a matrix-valued state with a limited number of memory slots. We approached this problem by framing it as an online optimization problem, deriving the memory's dynamic update rule from a single gradient descent step. The resulting recurrence features a state- and input-dependent gating mechanism, leading to an interpretable memory update process. A core feature of this mechanism is that each memory slot is updated exclusively with information that is orthogonal to its current state. This orthogonal update ensures that only new, non-redundant data is written into memory and minimize (reduce) the interference with previously stored information.

References

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

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

Arora, S., Eyuboglu, S., Zhang, M., Timalsina, A., Alberti, S., Zinsley, D., Zou, J., Rudra, A., and Ré, C. Simple linear attention language models balance the recall-throughput tradeoff. *arXiv preprint arXiv:2402.18668*, 2024.

Ba, J. L. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Beck, A. and Teboulle, M. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

Beck, M., Pöppel, K., Spanring, M., Auer, A., Prudnikova, O., Kopp, M., Klambauer, G., Brandstetter, J., and Hochreiter, S. xLSTM: Extended long short-term memory. *arXiv preprint arXiv:2405.04517*, 2024.

Behrouz, A., Zhong, P., and Mirrokni, V. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024.

Bisk, Y., Zellers, R., Gao, J., Choi, Y., 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.

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

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

Blelloch, G. E. Prefix sums and their applications. 1990.

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

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

Chen, B., Dao, T., Winsor, E., Song, Z., Rudra, A., and Ré, C. Scatterbrain: Unifying sparse and low-rank attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Chen, C., Chen, X., Ma, C., Liu, Z., and Liu, X. Gradient-based bi-level optimization for deep learning: A survey. *arXiv preprint arXiv:2207.11719*, 2022.

Choromanski, K., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J., Mohiuddin, A., Kaiser, L., et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

Clark, K., Guu, K., Chang, M.-W., Pasupat, P., Hinton, G., and Norouzi, M. Meta-learning fast weight language models. *arXiv preprint arXiv:2212.02475*, 2022.

Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

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

De, S., Smith, S. L., Fernando, A., Botev, A., Cristian-Muraru, G., Gu, A., Haroun, R., Berrada, L., Chen, Y., Srinivasan, S., et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024.

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

Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

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

Gonzalez, X., Warrington, A., Smith, J. T., and Linderman, S. W. Towards scalable and stable parallelization of nonlinear rnns. *arXiv preprint arXiv:2407.19115*, 2024.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. *Deep learning*, volume 1. MIT press Cambridge, 2016.

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

Gu, A., Dao, T., Ermon, S., Rudra, A., and Ré, C. HiPPO: Recurrent memory with optimal polynomial projections. *Advances in neural information processing systems*, 33: 1474–1487, 2020.

Gu, A., Goel, K., and Ré, C. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

Gu, A., Goel, K., Gupta, A., and Ré, C. On the parameterization and initialization of diagonal state space models. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K. (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=yJE7iQSAep.

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

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

Hua, W., Dai, Z., Liu, H., and Le, Q. Transformer quality in linear time. In *International conference on machine learning*, pp. 9099–9117. PMLR, 2022.

Irie, K., Schlag, I., Csordás, R., and Schmidhuber, J. Going beyond linear transformers with recurrent fast weight programmers. *Advances in neural information processing systems*, 34:7703–7717, 2021.

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

Karami, M. and Ghodsi, A. 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.

Karami, M., White, M., Schuurmans, D., and Szepesvári, C. Multi-view matrix factorization for linear dynamical system estimation. *Advances in Neural Information Processing Systems*, 30, 2017.

Karami, M., Schuurmans, D., Sohl-Dickstein, J., Dinh, L., and Duckworth, D. Invertible convolutional flow. *Advances in Neural Information Processing Systems*, 32, 2019.

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

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

Li, Y., Cai, T., Zhang, Y., Chen, D., and Dey, D. What makes convolutional models great on long sequence modeling? *arXiv preprint arXiv:2210.09298*, 2022.

Lim, Y. H., Zhu, Q., Selfridge, J., and Kasim, M. F. Parallelizing non-linear sequential models over the sequence length. *arXiv preprint arXiv:2309.12252*, 2023.

Liu, B., Ye, M., Wright, S., Stone, P., and Liu, Q. Bome! bilevel optimization made easy: A simple first-order approach. *Advances in neural information processing systems*, 35:17248–17262, 2022.

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

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

Mairal, J., Bach, F., Ponce, J., and Sapiro, G. Online learning for matrix factorization and sparse coding. *Journal of Machine Learning Research*, 11(1), 2010.

Mehta, H., Gupta, A., Cutkosky, A., and Neyshabur, B. Long range language modeling via gated state spaces. *arXiv preprint arXiv:2206.13947*, 2022.

Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gulcehre, C., Pascanu, R., and De, S. Resurrecting recurrent neural networks for long sequences. In *International Conference on Machine Learning*, pp. 26670–26698. PMLR, 2023.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The lambada dataset: Word prediction requiring a broad discourse context. *arXiv preprint arXiv:1606.06031*, 2016.

Parikh, N., Boyd, S., et al. Proximal algorithms. *Foundations and trends® in Optimization*, 1(3):127–239, 2014.

Poli, M., Massaroli, S., Nguyen, E., Fu, D. Y., Dao, T., Baccus, S., Bengio, Y., Ermon, S., and Ré, C. Hyena hierarchy: Towards larger convolutional language models. *International Conference on Machine Learning*, 2023.

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

Qin, Z., Han, X., Sun, W., Li, D., Kong, L., Barnes, N., and Zhong, Y. The devil in linear transformer. *arXiv preprint arXiv:2210.10340*, 2022.

Qin, Z., Yang, S., Sun, W., Shen, X., Li, D., Sun, W., and Zhong, Y. Hgrn2: Gated linear rnns with state expansion. *arXiv preprint arXiv:2404.07904*, 2024.

Romero, D. W., Kuzina, A., Bekkers, E. J., Tomczak, J. M., and Hoogendoorn, M. Ckconv: Continuous kernel convolution for sequential data. In *International Conference on Learning Representations*, 2021.

Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

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

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

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

Singhania, P., Singh, S., He, S., Feizi, S., and Bhatele, A. Loki: Low-rank keys for efficient sparse attention. *arXiv preprint arXiv:2406.02542*, 2024.

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

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

Strang, G. *Linear algebra and its applications*. 2000.

Sun, Y., Wang, X., Liu, Z., Miller, J., Efros, A., and Hardt, M. Test-time training with self-supervision for generalization under distribution shifts. In *International conference on machine learning*, pp. 9229–9248. PMLR, 2020.

Sun, Y., Li, X., Dalal, K., Xu, J., Vikram, A., Zhang, G., Dubois, Y., Chen, X., Wang, X., Koyejo, S., et al. Learning to (learn at test time): Rnns with expressive hidden states. *arXiv preprint arXiv:2407.04620*, 2024.

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6): 1–28, 2022.

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

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Von Der Malsburg, C. 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.

Wang, K. A., Shi, J., and Fox, E. B. Test-time regression: a unifying framework for designing sequence models with associative memory. *arXiv preprint arXiv:2501.12352*, 2025.

Wang, S., Li, B. Z., Khabsa, M., Fang, H., and Ma, H. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*, 2020.

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

Yang, S., Kautz, J., and Hatamizadeh, A. Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024a.

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

Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

Zhang, Y., Yang, S., Zhu, R., Zhang, Y., Cui, L., Wang, Y., Wang, B., Shi, F., Wang, B., Bi, W., 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 concepts 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) 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 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.

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 11) and state decay (Equation 15). 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 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 forms 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.

Remark A.1 (Parallel and Hardware Efficient Implementation). *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 another approach, Sun et al. (2024) introduced a parallel chunkwise solution using mini-batch gradient descent. This method divides a sequence into chunks and utilizes the state at the beginning of each chunk to compute the gradients for all time steps within that chunk in parallel. Exploring parallel solutions for our proposed non-linear orthogonal state recurrence remains a promising direction for future work.*

Remark A.2 (Delta Rule). *Removing the non-linearity $\phi(\cdot)$ from the compression layer simplifies the online gradient descent update rule in (1, 2) to*

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \gamma_t(\mathbf{S}_{t-1}\boldsymbol{k}_t - \boldsymbol{v}_t)\boldsymbol{k}_t^\top = \mathbf{S}_{t-1}(\mathbf{I} - \gamma_t \boldsymbol{k}_t\boldsymbol{k}_t^\top) + \gamma_t \boldsymbol{v}_t\boldsymbol{k}_t^\top \tag{12}$$

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., 2024b). 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.

Summary of contributions In this paper, we propose Lattice, a novel approach designed to addresses 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 in an online optimization framework. This approach allows us to derive efficient recursive update rules for the memory (representing K-V associations) based on its existing state and the current token, resulting in sub-quadratic complexity. In contrast to existing SSMs/RNNs, which often rely on heuristics for memory management and lack explicit optimization for compression, we formulate the compression task as an optimization problem and use online gradient descent to drive the recurrent update rule for the memory, which results in an interpretable and expressive non-linear recurrent model. Lattice updates each memory slot exclusively with non-redundant information, specifically by incorporating only the component of the input token that is orthogonal to the current state of that memory slot.

B. Background

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

$$\boldsymbol{y}_t = \mathcal{V}_t \, \texttt{Softmax}(\mathcal{K}_t^\top \, \boldsymbol{q}_t) \, . \tag{13}$$

Here, the queries, keys, and values are computed by linear projections of the input: $\boldsymbol{q}_t = \mathbf{W}_q \, \boldsymbol{x}_t$, $\boldsymbol{k}_t = \mathbf{W}_k \, \boldsymbol{x}_t$, $\boldsymbol{v}_t = \mathbf{W}_v \, \boldsymbol{x}_t$, where $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{d\times d}$ are learnable weight matrices. The key-value memory, represented by the caches $\mathcal{K}_t \in \mathbb{R}^{d\times t}$ and $\mathcal{V}_t \in \mathbb{R}^{d\times t}$, stacks the key and value vectors of each new token, leading to linearly growing caches. 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 $\boldsymbol{a}_t = \texttt{Softmax}(\mathcal{K}_t^\top \, \boldsymbol{q}_t) \in \mathbb{R}^t$ Here, the vector $\boldsymbol{a}_t \in \mathbb{R}^t$ is the collection of the attention scores capturing correlations between t-th token and its historic context (past tokens). Hence, the attention in equation 13 can be seen as a non-linear query from an unbounded memory. The key-value cache size growth poses a significant memory bottleneck during inference, especially for long sequences. Additionally, each retrieval operation scales as linearly with sequence length, resulting in an overall quadratic computational complexity $\mathcal{O}(T^2)$ for generating a full sequence of length T.

To address the computational and memory bottleneck 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 with a dot product of feature maps, $\phi(\boldsymbol{q}_t)$, $\phi(\boldsymbol{k}_t)$, (Katharopoulos et al., 2020), commonly known as *linear attention* (LA). The linear attention can be expressed 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 following linear recurrent model, also known as input dependent state-space model (SSM) [4]:

$$\{\boldsymbol{y}_t\}_{t=1}^T = \mathrm{LA}(\{\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t\}_{t=1}^T) := \begin{cases} \mathbf{S}_t = \mathbf{S}_{t-1} + \boldsymbol{v}_t \, \boldsymbol{k}_t^\top, & \textit{recurrence} \\ \boldsymbol{y}_t = \mathbf{S}_t \boldsymbol{q}_t & \textit{memory read-out} \end{cases} \tag{14}$$

This representation employs a simple linear recurrence to update the matrix-valued state $\mathbf{S}_t$, which compactly stores 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 constant-time complexity per token at the inference.

Another approach to maintain bounded computational and memory requirements is to maintain a fixed-size key-value cache, where the memory matrices $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{m\times d}$ are constrained to a fixed length $m \ll T$. A simple implementation of this idea is the sliding window attention which retains the most recent m tokens by maintaining a first-in-first-out (FIFO) queue. While computationally efficient, sliding window attention suffers from a limited receptive field. This restricts the model's

[4] As with many linear attention models, the normalization term, which can cause numerical instabilities (Qin et al., 2022), is dropped here. Furthermore, identity mapping is used as the feature map, effectively absorbing any transformation into the corresponding projection layers.

Table 3: Comparison of the objective functions and their corresponding online gradient descent updates for the proposed and existing RNNs. We include several linear RNNs for comparison: Linear-Attention(LA) (Katharopoulos et al., 2020), Mamba2 (Dao & Gu, 2024) and DeltaNet (Schlag et al., 2021; Yang et al., 2024b), Gated-DeltaNet (Yang et al., 2024a), and TTT (Sun et al., 2024) It is worth noting that, after re-scaling, the effective recurrent update of the proposed RNNs becomes $\mathbf{S}_t = \mathbf{1}\,\boldsymbol{\beta}_t^\top \odot \big(\mathbf{S}_{t-1} + \Delta\mathbf{S}_t\big)$(*equation* 11). LA can be interpreted as online gradient descent with a fixed step size ($\gamma_t = 1$); however, more flexible, 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 performing online gradient descent with L2 regularization and regularization factor 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). Here, $\times_1$ denotes vector-tensor product defined as $\boldsymbol{e}^\top \times_1 \big[\mathbf{J}_1,\ \ldots, \mathbf{J}_m\big] = \big[\boldsymbol{e}^\top\mathbf{J}_1,\ \ldots, \boldsymbol{e}^\top\mathbf{J}_m\big]$.

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	$-\langle \mathbf{S}_t\boldsymbol{k}_t, \boldsymbol{v}_t\rangle + \frac{\lambda_t}{2}\|\mathbf{S}_t\|_2^2$	$\mathbf{S}_t = \mu_t\mathbf{S}_{t-1} + \boldsymbol{v}_t\boldsymbol{k}_t^\top$
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	$\Big\|\mathbf{S}_t\boldsymbol{k}_t - \boldsymbol{v}_t\Big\|^2 + \frac{\lambda_t}{2}\|\mathbf{S}_t\|_2^2$	$\mathbf{S}_t = \mu_t\mathbf{S}_{t-1}(\mathbf{I} - \gamma_t\boldsymbol{k}_t\boldsymbol{k}_t^T) + \gamma_t\boldsymbol{v}_t\boldsymbol{k}_t^T$
TTT	$\Big\|\phi(\mathbf{S}_t\boldsymbol{k}_t) - \boldsymbol{v}_t\Big\|^2$	$\mathbf{S}_t = \mathbf{S}_{t-1} - \gamma_t\boldsymbol{e}_t^\top \frac{\mathbf{P}(\boldsymbol{z}_t)}{\|\boldsymbol{z}_t\|}\boldsymbol{k}_t^\top$
Lattice (Dec) (3)	$\Big\|\phi(\mathbf{S}_t)\boldsymbol{k}_t - \boldsymbol{v}_t\Big\|^2$	$\mathbf{S}_t = \mathbf{S}_{t-1} - \gamma_t\boldsymbol{e}_t^\top \times_1 \Big[\frac{\mathbf{P}(\boldsymbol{s}_1)}{\|\mathbf{s}_1\|}, \ldots, \frac{\mathbf{P}(\boldsymbol{s}_m)}{\|\mathbf{s}_m\|}\Big] \odot \boldsymbol{k}_t^\top$
Lattice (Enc) (6)	$\Big\|\phi(\mathbf{S}_t)^\top\boldsymbol{v}_t - \boldsymbol{k}_t\Big\|^2$	$\mathbf{S}_t = \mathbf{S}_{t-1} - \gamma_t\boldsymbol{v}_t^\top \times_1 \Big[\frac{\mathbf{P}(\boldsymbol{s}_1)}{\|\mathbf{s}_1\|}, \ldots, \frac{\mathbf{P}(\boldsymbol{s}_m)}{\|\mathbf{s}_m\|}\Big] \odot \boldsymbol{e}_t^\top$
Lattice (Sim) (8)	$-\langle \phi(\mathbf{S}_t)^\top\boldsymbol{v}_t,\ \boldsymbol{k}_t\rangle$	$\mathbf{S}_t = \mathbf{S}_{t-1} + \gamma_t\boldsymbol{v}_t^\top \times_1 \Big[\frac{\mathbf{P}(\boldsymbol{s}_1)}{\|\mathbf{s}_1\|}, \ldots, \frac{\mathbf{P}(\boldsymbol{s}_m)}{\|\mathbf{s}_m\|}\Big] \odot \boldsymbol{k}_t^\top$

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 often exhibit structured low-rank (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 distill and store the essential context while discarding less relevant or redundant information.

B.1. 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 $\ell 2$ regularization to the state matrix $\mathbf{S}_t$ yields the regularized objective function: $\hat{\mathcal{L}}_t = \|\phi(\mathbf{S}_t)\,\boldsymbol{k}_t - \boldsymbol{v}_t\|_F^2 + \frac{\lambda_t}{2}\|\mathbf{S}_t\|^2$, 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 3):

$$\mathbf{S}_t = \mu_t\mathbf{S}_{t-1} - \gamma_t\nabla_S\mathcal{L}(\mathbf{S}_{t-1}, \boldsymbol{v}_t, \boldsymbol{k}_t), \tag{15}$$

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.

Alternative to $\ell 2$ regularization, we can induce sparsity in the memory states by applying element-wise $\ell_1$norm [5], resulting in the following objective function:

$$\hat{\mathcal{L}}_t = \|\phi(\mathbf{S}_t)\,\boldsymbol{k}_t - \boldsymbol{v}_t\|^2 + \lambda_t\|\mathbf{S}_t\|_1. \tag{16}$$

This non-differentiable composite objective can be efficiently optimized using the *Proximal Gradient Descent Algorithm*, which iteratively performs a gradient descent with the smooth component and then applies the proximal operator associated with the non-differentiable regularizer (Parikh et al., 2014). This iterative procedure, commonly known as the Iterative Shrinkage-Thresholding Algorithm (ISTA) (Parikh et al., 2014; Beck & Teboulle, 2009), yields the update rule :

$$\mathbf{S}_t = \mathrm{prox}_{\gamma_t\lambda_t\|\cdot\|_1}\left(\mathbf{S}_{t-1} - \gamma_t\nabla_S\|\phi(\mathbf{S}_t)\,\boldsymbol{k}_t - \boldsymbol{v}_t\|^2\right), \tag{17}$$

where the proximal operator for the $\ell 1$ norm corresponds to the *shrinkage (soft thresholding) operation*, defined as: $\mathrm{prox}_{\mu_t\|\cdot\|_1}(x) = \mathrm{sign}(x)\max(|x| - \mu_t, 0)$. By suppressing small values, this recurrence promotes sparsity in the learned memory representations.

[5] The $\ell 1$ norm is a relaxed version of the hard sparsity constrain, which drives small states toward zero.

B.2. Computational Complexity

The update rules presented in this work (equation 10) involve computing the projection $\boldsymbol{h}_t^{\perp \boldsymbol{s}_i} = \mathbf{P}\boldsymbol{h}_t$. Given its identity matrix plus rank-one form, the projection operation reduces to a dot product and a scalar-vector multiplication:

$$\boldsymbol{h}_t^{\perp \boldsymbol{s}_i} = \boldsymbol{h}_t - \frac{\boldsymbol{s}_i(\boldsymbol{s}_i^\top \boldsymbol{h}_t)}{\|\boldsymbol{s}_i\|^2},$$

avoiding a full matrix-vector multiplication. Therefore, the computational cost of each projection is linear in the state dimensions leading to an overall complexity of $\mathcal{O}(d\,m)$ for the recurrence in equation 10. Furthermore, by eliminating the need for vector-Jacobian-product (`vjp`) computations in the general gradient descent update rule (equation 2), this explicit form leads to a more efficient and scalable implementation.

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.[6] In this case, the decoding function becomes: $\hat{\boldsymbol{v}}_t = g(\boldsymbol{k}_t; \mathbf{S}_t) = \phi(\mathbf{S}_t\boldsymbol{k}_t)$. 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:

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

Subsequently, the gradient descent update follows a nonlinear recurrence:

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \gamma_t \boldsymbol{e}_t^\top \frac{\mathbf{P}(\boldsymbol{z}_t)}{\|\boldsymbol{z}_t\|}\boldsymbol{k}_t^\top \tag{18}$$

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 $\hat{\boldsymbol{v}}_t$.

Remark B.1. *The concept of normalizing the state vectors in our compression model, as described in §2.1, 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.*

C. Proofs

C.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:

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

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

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:

$$\begin{aligned} \nabla_{\mathbf{S}} \mathcal{L}_t &= \left[\frac{2\left(\boldsymbol{\phi}_1^\top \boldsymbol{v}_t - (\boldsymbol{k}_t)_1\right)}{\|\boldsymbol{s}_1\|} \left(\mathbf{I}_d - \boldsymbol{\phi}_1 \boldsymbol{\phi}_1^\top\right) \boldsymbol{v}_t,\ \cdots,\ \frac{2\left(\boldsymbol{\phi}_m^\top \boldsymbol{v}_t - (\boldsymbol{k}_t)_m\right)}{\|\boldsymbol{s}_m\|} \left(\mathbf{I}_d - \boldsymbol{\phi}_m \boldsymbol{\phi}_m^\top\right) \boldsymbol{v}_t\right] \qquad (19) \\ &= \left[\frac{2\,\boldsymbol{e}_1}{\|\boldsymbol{s}_1\|} \mathbf{P}(\boldsymbol{s}_1)\, \boldsymbol{v}_t,\ \cdots,\ \frac{2\,\boldsymbol{e}_m}{\|\boldsymbol{s}_m\|} \mathbf{P}(\boldsymbol{s}_m)\, \boldsymbol{v}_t\right] \qquad \square \end{aligned}$$

Proof of Proposition 2.1. Consider the unit sphere $\mathcal{C} = \{\boldsymbol{s} \in \mathbb{R}^d \mid \|\boldsymbol{s}\| = 1\}$ which is a 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 constructed such that it 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

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

where $\exp_{\mathbf{s}}$ 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, which 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 11) is equivalent to performing a Riemannian gradient descent step with retraction on the manifold $\mathcal{C}$. $\square$ $\square$

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: *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

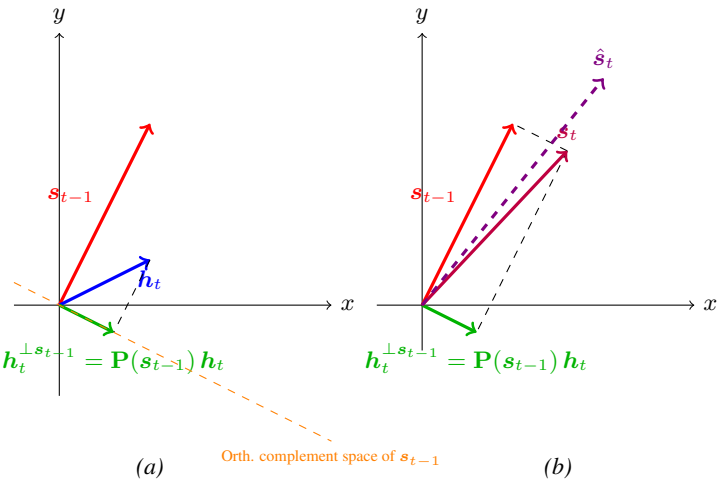

Figure 3: An illustration of the proposed update rule. (a) Example of a single memory slot state, $\boldsymbol{s}_t$, an incoming token representation, $\boldsymbol{h}_t$, and its component orthogonal to the current state, $\boldsymbol{h}_t^{\perp \boldsymbol{s}_{t-1}}$. (b) The updated state according 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.

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., 2024b), Gated DeltaNet (Yang et al., 2024a), Mamba2 (Dao & Gu, 2024). As discussed in the paper, the Lattice layers incorporate $\ell 2$-normalization on the state, whereas TTT applies $\ell 2$-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 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., 2017). The model architecture used for Lattice is illustrated in Figure 4.

Furthermore, the performance of the trained models on various zero-shot common sense reasoning tasks —including LAMBADA (LMB.) (Paperno et al., 2016), 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) commonly used for benchmarking (Zhang et al., 2024; Yang et al., 2024b)- are reported in table Table 1. As the results show, Lattice outperforms all the baseline models on these benchmarks, achieving the highest average accuracy.

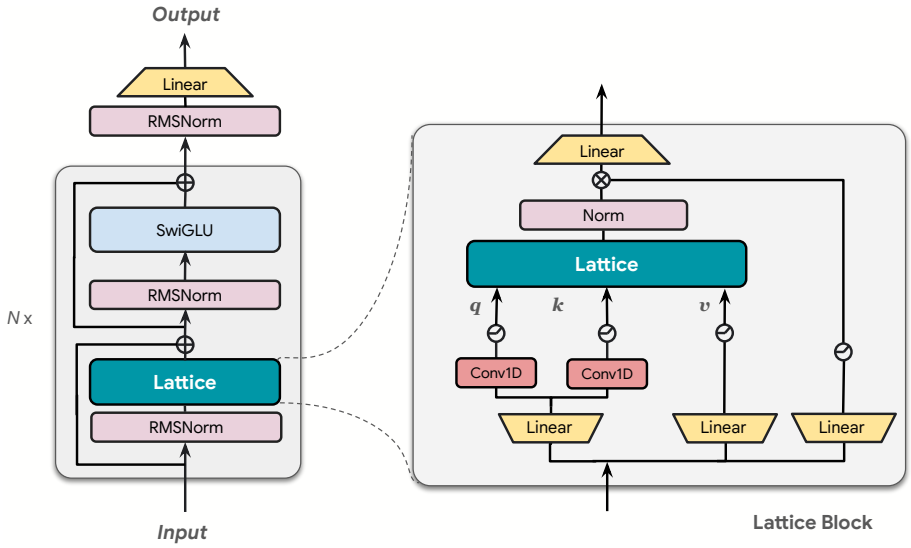

Figure 4: *(Left)* Block diagram of the language model. *(Right)* The Lattice block. Following the architecture used in Mamba (Gu & Dao, 2023), each sequence mixing block is composed of a pair of short `Conv1D` for the pair $\{q, k\}$ and the Lattice is followed by a `GeLU` post-gate.

Table 4: Performance comparison of language models of size 110M parameters trained on datasets: the Pile (with context length of 2k and 8k tokens) and Books (with various context length), Baselines include: Transformer++ (Touvron et al., 2023), Linear-Attention(LA) (Katharopoulos et al., 2020), DeltaNet (Yang et al., 2024b), Gated-DeltaNet (Yang et al., 2024a), Mamba2 (Dao & Gu, 2024), and TTT (Sun et al., 2024). The best results are highlighted.

Model	Pile (2k) ppl ↓	Pile (8k) ppl ↓	Books (512) ppl ↓	Books (1k) ppl ↓	Books (2k) ppl ↓	Books (4k) ppl ↓	Books (8k) ppl ↓	Books (16k) ppl ↓
110M params / 2.4B tokens								
Transformer++	11.58	11.75	20.60	19.39	18.89	18.38	18.85	29.41
Linear-Attention	12.56	13.63	21.04	20.18	19.82	19.69	20.34	21.86
Mamba2	11.51	11.42	19.94	18.90	18.34	18.07	18.23	18.48
DeltaNet	11.62	11.40	20.28	19.11	18.33	17.90	18.05	18.12
Gated-DeltaNet	11.31	11.04	19.76	18.60	18.00	17.48	17.40	17.49
TTT	11.59	11.45	20.11	19.03	18.36	18.03	18.05	18.46
Lattice-DEC (4)	10.88	10.51	19.06	17.90	17.14	16.72	16.62	16.97
Lattice-ENC (7)	10.90	10.57	19.11	18.01	17.22	16.80	16.66	
Lattice-SIM (9)	10.89	10.66	19.08	17.94	17.23	16.72	16.73	16.82