FORWARD ONLINE LEARNING DEEP NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern deep neural networks are typically trained offline via computationally intensive large-scale backpropagation on massive datasets. In deployment scenarios, adapting the model weights to new data streams again requires costly backpropagation. In this paper, we derive from first principles a novel deep architecture, which we call FOLD, that supports both efficient *offline training via backward propagation* and *online learning via forward propagation*. Each layer consists of *a novel linear-time operator* that can replace both the attention and the MLP operators in a Transformer. The weights of each such operator are essentially a learned dictionary for its input data; in this way online adaptation to incoming new data becomes an online dictionary learning problem. As a result, the model weights can be updated at test time through *forward propagation as an optimization step* to better align the dictionaries with every new sample sequence. Experiments on synthetic and real-world benchmarks show that this new architecture performs competitively against the state-of-the-art linear-time recurrent deep architectures such as Gated DeltaNet with back propagation training while demonstrating qualitatively superior online learning capabilities with its forward propagation mechanism. We also show that incorporating explicit forward propagation into the architecture has side benefits, such as strong long-context performance with only subquadratic operations. Our results call into question the necessity of solely using back propagation to adapt pretrained models.

1 INTRODUCTION

The Transformer architecture has led to breakthroughs across numerous applications in machine learning, computer vision, and natural language processing (Vaswani et al., 2017; Brown et al., 2020; Dosovitskiy et al., 2021), owing largely to its self-attention operator. However, two of its limitations have become increasingly evident as applications move towards problem domains with vastly longer contexts. First, the computational and memory complexity of the attention operator scales quadratically with the number of input tokens, leading to prohibitive training and inference costs. Second, like other modern deep networks, transformers are trained offline on massive datasets via back propagation (BP) and have their weights frozen upon deployment, so that a trained model processes a sequence without learning anything from it. Adaptation to new distributions then requires finetuning the model with BP, which grows more expensive as models scale.

To address the first limitation, researchers have developed linear attention (Katharopoulos et al., 2020), which replaces regular softmax-based attention with a recurrent state, also known as *fast weight*, whose complexity scales linearly with the context length. Early variants of linear attention (Peng et al., 2023a; Gu & Dao, 2023) often underperform conventional softmax attention, as the fast weight accumulates key-value pairs indiscriminately. Later works further introduce test-time training (TTT) (Sun et al., 2024; Yang et al., 2024; 2025a), a framework that formulates the recurrent state update as an online regression problem that learns a mapping between the key-value pairs. The TTT framework has demonstrated much improved long-context modeling and retrieval capabilities, but the model parameters remain frozen after training, leaving the second limitation unaddressed.

The ability to enable trained models to incrementally adapt and acquire new knowledge, also known as continual learning, has been a long-standing problem, as naively training on new data leads to *catastrophic forgetting*. Recent works address the challenge primarily through regularized self-supervised learning objectives, data replay, and expanding architectures (cf. Tong et al., 2022; Peng et al., 2023b; Yang et al., 2025b, and references therein), but still rely on (end-to-end) BP as the learning mechanism. A less explored alternative is to train model parameters in the *forward pass* via local update rules. Indeed, a classical example of such an approach is Oja's rule for online PCA (Oja,

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.

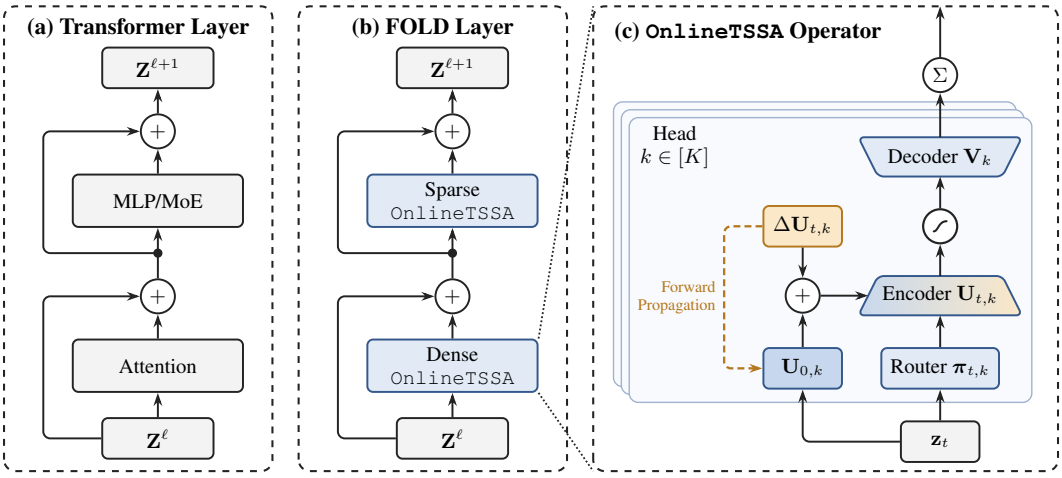


Figure 1: **The Forward Online Learning Deep (FOLD) architecture.** Different from the Transformer layer (a) which has distinct attention and MLP/MoE operators, a FOLD layer (b) adopts a densely and a sparsely routed `OnlineTSSA` operator (c). Within each head, `OnlineTSSA` encodes the token $\mathbf{z}_t$ with the dictionary $\mathbf{U}_{t,k}$ obtained as the sum of a slow base $\mathbf{U}_{0,k}$ and a fast correction $\Delta\mathbf{U}_{t,k}$. Notably, $\Delta\mathbf{U}_{t,k}$ (10) is updated online by the prediction residual (8) of each token, which can be consolidated into $\mathbf{U}_{0,k}$ at the end of each sequence (13), i.e., forward propagation.

1982), and more recent work attempts to build deep networks using forward learning rules (Chan et al., 2015; Wu et al., 2021; Chan et al., 2022; Hinton, 2022). These methods avoid expensive backward passes, but typically assume simplistic data models and have so far only demonstrated limited performance at small scale.

In this work, we address both limitations by adopting the same design principles behind performant white-box architectures (Yu et al., 2023; Wu et al., 2024) for continual online learning. These earlier works have shown that unrolling iterative optimization algorithms for a principled objective of *information gain*, such as maximal coding rate reduction (Yu et al., 2020), yields efficient transformer-like architectures whose weights take on explicit roles as dictionaries (or subspace bases) for the data distribution. In particular, Token Statistics Self-Attention (`TSSA`) proposed in Wu et al. (2024) alleviates the quadratic cost and scales linearly with the number of input tokens. Whereas prior work learns these dictionaries offline via back propagation and holds them fixed thereafter, our key insight in this work is that the learned dictionaries can be made adaptive to incoming tokens, naturally leading to an online dictionary learning problem. More concretely, each optimization step of the learning objective corrects the dictionary in the direction of minimizing the reconstruction error of each incoming token. Because the correction applies to the dictionary itself rather than to a separate recurrent state as in TTT, it can be accumulated over a sequence and naturally consolidated online into the pretrained model weights. We call the resulting operator `OnlineTSSA`.

Somewhat surprisingly, we find that the so-derived `OnlineTSSA` operator is *the only basic operator* needed to build a Forward Online Learning Deep (FOLD) architecture, replacing not only the attention operator in a Transformer, but also the multi-layer perceptron (MLP) or the mixture of experts (MoE) operator (Shazeer et al., 2017). The only difference between the two `OnlineTSSA` operators in the same layer is how a token is routed to a head: the router for the second operator is much sparser than that for the first. The resulting architecture is causal, scales linearly with context length, and can be learned via *both offline pretraining via back propagation and online learning via forward propagation*. Figure 1 shows a layer of the new architecture next to that of a Transformer. We develop theoretical and experimental justification for our architecture below, and we here summarize the main technical contributions:

- We show how to adapt the white-box architecture design principles to the continual online learning setting, which adapts the linear-time attention operator `TSSA` to an `OnlineTSSA`, whose recurrent state is a correction to the model's own weights (as a learned dictionary).
- We show that the basic dictionary learning unit for subspace attention can be used for both attention and MLP modules in a transformer, yielding a deep architecture consisting of homogeneous units that support both backward and forward learning.

- We validate our proposed architecture on both synthetic tasks and causal language modeling, showcasing both its effectiveness in long-context modeling and forward learning capabilities.

2 Preliminaries and Background

Standard self-attention in Transformer architectures computes sequence representations through dynamic token interactions. Given an input sequence of n tokens, $\mathbf{Z} = [\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_n] \in \mathbb{R}^{d\times n}$, the output at each position is formed as a linear combination of value projections across the sequence, weighted by softmax-normalized query-key affinity scores. Formally, this mechanism exhibits a *self-expressive* property: each output token is constructed as a linear combination of other tokens within the same sequence (Yu et al., 2020; Vidal, 2021). This can be interpreted through the lens of dictionary learning, where the token sequence $\mathbf{Z}$ itself serves as a data-dependent dictionary used to construct each output token as a linear combination of the other tokens. While highly expressive in terms of adapting the model to the structure of a particular input sequence, this is also a root cause of the well-known quadratic sequence length computational complexity ($\mathcal{O}(n^2)$) of standard Transformers. Because the implicit dictionary and its associated encoding are given directly as sequence tokens, the effective size of the dictionary expands with sequence length, necessitating $n \times n$ pairwise evaluations.

A potential alternative to avoid an expanding, sequence-dependent dictionary is to instead model the dictionary representation explicitly rather than constructing the dictionary implicitly from the arriving sequence. However, fixing a dictionary also potentially sacrifices model expressiveness, which can be partially mitigated by using a mixture of subspaces (MoS) framework to allow for multiple potential dictionaries. Under a MoS model[1], the feature space is partitioned into K low-dimensional linear subspaces in $\mathbb{R}^d$, the k-th of which is spanned by the columns of a matrix $\mathbf{U}_k \in \mathbb{R}^{d\times p}$. This gives a token reconstruction of the form:

$$\mathbf{z}_j \approx \sum_{k=1}^{K} \boldsymbol{\pi}_{j,k} \mathbf{U}_k \mathbf{a}_{j,k}, \tag{1}$$

where the columns of $\mathbf{U}_k$ are the atoms of the k-th dictionary, $\mathbf{a}_{j,k} = \mathbf{U}_k^\top \mathbf{z}_j \in \mathbb{R}^p$ is the corresponding code within the k-th subspace, and $\boldsymbol{\pi}_{j,k}$ gives the mixture weighting of token j in the k-th subspace. By replacing sequence-dependent pairwise tokens with this fixed-capacity subspace model, the computational complexity decouples from sequence length n, yielding linear computational scaling with sequence length. However, even taking a mixture of multiple subspaces still might struggle to capture the expressive power of standard self-attention. Below we derive a model which seeks to both maintain the sub-quadratic computational complexity of explicit dictionary representations while also rapidly adapting the dictionary to the incoming sequence as in regular self-attention.

2.1 Subspace Compression via MCR^2 and TSSA

Prior work has proposed so-called 'white-box' architectures from the principle of Maximal Coding Rate Reduction (MCR^2) (Yu et al., 2023; Wu et al., 2024) by leveraging this perspective to derive Transformer-style operators by optimizing a rate reduction metric. Specifically, the MCR^2 objective (Yu et al., 2020) learns representations of data with a mixture of low-dimensional linear subspaces by maximizing the difference between the coding rate (to precision ϵ) of the entire feature space ($R(\mathbf{Z})$) and the sum of the coding rates of the individual subspaces ($R_c(\mathbf{Z}, \boldsymbol{\Pi})$).

$$\Delta R(\mathbf{Z}, \boldsymbol{\Pi}) \doteq \underbrace{\frac{1}{2}\log\det\left(\mathbf{I} + \frac{d}{\epsilon^2}\frac{1}{n}\mathbf{Z}\mathbf{Z}^\top\right)}_{\doteq R(\mathbf{Z})} - \underbrace{\frac{1}{2}\sum_{k=1}^{K}\frac{n_k}{n}\log\det\left(\mathbf{I} + \frac{d}{\epsilon^2}\frac{1}{n_k}\mathbf{Z}\mathrm{Diag}(\boldsymbol{\pi}_k)\mathbf{Z}^\top\right)}_{\doteq R_c(\mathbf{Z},\boldsymbol{\Pi})}, \tag{2}$$

where $\mathrm{Diag}(\mathbf{v})$ denotes placing the entries of $\mathbf{v}$ along the diagonal of a matrix, $\boldsymbol{\pi}_k \in \mathbb{R}^n_+$ is a vector whose entries weigh the membership of each token within the k-th subspace, and we define constants $n_k = \langle \boldsymbol{\pi}_k, \mathbf{1}\rangle$, $\alpha_k = d/(\epsilon^2 n_k)$, and $\alpha = d/(\epsilon^2 n)$.

From this, the authors of Wu et al. (2024) derive a variational bound to the R_c term of the MCR^2 objective and, following Yu et al. (2023), propose an attention operator by unrolling the gradient of

[1] Sparse coding is a canonical instance of a mixture of subspaces model, where encoding a signal with p elements from an over-complete dictionary of M atoms gives a combinatorial number $\binom{M}{p}$ of linear subspaces.

this variational bound, giving the *Token Statistics Self-Attention* (TSSA) operator:

$$\nabla_{\mathbf{Z}} R^{\text{var}}_{c,f}(\mathbf{Z}, \mathbf{\Pi} \mid \{\mathbf{U}_k\}_{k=1}^K) = \frac{1}{n}\sum_{k=1}^K \mathbf{U}_k \underbrace{\text{Diag}\left(\nabla f\left[(\mathbf{U}_k^\top \mathbf{Z})^{\odot 2}\frac{\boldsymbol{\pi}_k}{\langle \boldsymbol{\pi}_k, \mathbf{1}\rangle}\right]\right)}_{\doteq \mathbf{D}(\mathbf{Z}, \boldsymbol{\pi}_k \mid \mathbf{U}_k)} \mathbf{U}_k^\top \mathbf{Z}\text{Diag}(\boldsymbol{\pi}_k) \tag{3}$$

$$\mathbf{z}_j^+ = \mathbf{z}_j - \tau \nabla_{\mathbf{z}_j} R^{\text{var}}_{c,f}(\mathbf{Z}, \mathbf{\Pi} \mid \{\mathbf{U}_k\}_{k=1}^K) = \mathbf{z}_j \underbrace{-\frac{\tau}{n}\sum_{k=1}^K \boldsymbol{\pi}_{j,k}\mathbf{U}_k \mathbf{D}(\mathbf{Z}, \boldsymbol{\pi}_k \mid \mathbf{U}_k)\mathbf{U}_k^\top \mathbf{z}_j}_{\texttt{TSSA}(\mathbf{Z} \mid \{\mathbf{U}_k\}_{k=1}^K)} \tag{4}$$

where $\nabla f[\cdot]$ applies the gradient of $f(x) = \log(1 + (d/\epsilon^2)x)$ entry-wise, $(\cdot)^{\odot 2}$ denotes entry-wise squaring, and τ is a fixed step size in the unrolled architecture. Wu et al. (2024) follow Yu et al. (2023) in assigning weightings to the various subspaces via

$$\boldsymbol{\pi}_{j,k} = \frac{\exp\left(\|\mathbf{U}_k^\top \mathbf{z}_j\|_2^2/2\sigma^2\right)}{\sum_{l=1}^K \exp\left(\|\mathbf{U}_l^\top \mathbf{z}_j\|_2^2/2\sigma^2\right)}, \qquad \forall j \in [n],\ k \in [K], \tag{5}$$

with σ being a fixed temperature scaling, which can be shown to be an optimal posterior of subspace membership under a mixture of Gaussian subspaces model (Yu et al., 2023).

From a dictionary learning perspective, if we let $\mathbf{a}_{j,k} \doteq \mathbf{U}_k^\top \mathbf{z}_j \in \mathbb{R}^p$ be interpreted as the code of $\mathbf{z}_j$ under the k-th dictionary and $\mathbf{A}_k = [\mathbf{a}_{1,k}, \ldots, \mathbf{a}_{n,k}] \in \mathbb{R}^{p\times n}$ as the collections of codes across the sequence, (4) becomes

$$\mathbf{z}_j^+ = \mathbf{z}_j - \frac{\tau}{n}\sum_{k=1}^K \boldsymbol{\pi}_{j,k}\mathbf{U}_k \text{Diag}\left(\nabla f\left[\mathbf{A}_k^{\odot 2}\frac{\boldsymbol{\pi}_k}{\langle \boldsymbol{\pi}_k, \mathbf{1}\rangle}\right]\right)\mathbf{a}_{j,k}. \tag{6}$$

Thus, each branch/head of the operator encodes the token into the k-th dictionary, applies a diagonal gate determined by the routed code statistics, and decodes the gated code back to the token space; the router then combines the branch contributions via the weighted sum in $\mathbf{\Pi} = [\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_K] \in \mathbb{R}^{n\times K}$. In short, TSSA is a routed encode–gate–decode operator whose weights are exactly the dictionaries $\{\mathbf{U}_k\}_{k=1}^K$. Each branch therefore has a structure similar to a gated MLP such as SwiGLU (Shazeer, 2020), $\mathbf{W}_{\text{down}}\big(\text{SiLU}(\mathbf{W}_{\text{gate}}\mathbf{z}_j) \odot \mathbf{W}_{\text{up}}\mathbf{z}_j\big)$, with two differences: the three projections are tied to the dictionary $\mathbf{U}_k$, and the gate is computed from the routed statistics of the codes across the sequence rather than from the token alone. The latter is how TSSA mixes information across tokens. The routed sum over branches likewise resembles an MoE layer whose experts are gated MLPs.

Interleaving TSSA and MLP layers (similar to Transformers) leads to a neural architecture called the Token Statistics Transformer (ToST), which has been shown to be competitive in various vision and language tasks. The authors of Wu et al. (2024) apply a causal mask and learned positional encoding to extend ToST to sequence tasks such as causal language modeling; however, a notable drawback of this formulation is that the dictionaries $\{\mathbf{U}_k\}_{k=1}^K$ are learned offline and then held constant during inference. A dictionary fit to the training distribution has no reason to remain aligned with the tokens of any particular sequence. In fact, the variational bound used to derive TSSA only becomes tight when the $\mathbf{U}_k$ dictionary aligns with the eigen-space of the tokens that have been assigned to subspace k. As a result, in a causal prediction task, one would like the dictionaries to be adapted on-the-fly to match each new sequence of tokens. This motivates alternative network design choices which seek to adapt the subspace dictionaries to the statistics of the incoming tokens assigned to that subspace. This allows for the sub-quadratic computational benefits of maintaining a fixed-dimension dictionary within each subspace, yet still rapidly adapting the dictionary to the sequence at inference to match the expressive power of standard self-attention.

3 Forward Online Learning for TSSA

To develop the specifics of our model, note that in TSSA, the dictionaries $\{\mathbf{U}_k\}_{k=1}^K$ are learned offline by backpropagation and then remain frozen after training. Our central idea is to retain each learned dictionary as a *base* dictionary while adding a recurrent correction that is updated as tokens arrive. The correction then aligns the effective dictionary with the current sequence without discarding the representation learned during training.

3.1 Online Dictionary Adaptation

Here we focus our development on processing a sequence causally, one token at a time. To distinguish this setting from the generic sequence $\mathbf{Z} \in \mathbb{R}^{d\times n}$ of Section 2, we write $\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_T] \in \mathbb{R}^{d\times T}$, where $\mathbf{z}_t \in \mathbb{R}^d$ denotes the token arriving at time t. We let $\mathbf{U}_{0,k} \in \mathbb{R}^{d\times p}$ denote the k-th base dictionary, which will serve as the initialization of our dictionary for each sequence, and $\mathbf{a}_{t,k} = \mathbf{U}_{0,k}^\top \mathbf{z}_t \in \mathbb{R}^p$ the code of $\mathbf{z}_t$ under this dictionary. A router assigns subspace weightings/probabilities $\boldsymbol{\pi}_{t,k}$, similar to those in (5), to each token such that $\boldsymbol{\pi}_{t,k} \geq 0$ and $\sum_{k=1}^K \boldsymbol{\pi}_{t,k} = 1$. We give more details about the practical implementation of the router in Appendix C.

Online reconstruction objective. The base dictionaries $\{\mathbf{U}_{0,k}\}_{k=1}^K$ encode structure learned offline and may not represent the tokens of the current sequence well. We therefore pair each base dictionary with a fast correction $\Delta\mathbf{U}_{t,k} \in \mathbb{R}^{d\times p}$, initialized at $\Delta\mathbf{U}_{0,k} = \mathbf{0}$, and reconstruct with the effective dictionary $\mathbf{U}_{t,k} \doteq \mathbf{U}_{0,k} + \Delta\mathbf{U}_{t,k}$. Following the mixture model (1), the reconstruction error of $\mathbf{z}_t$ is measured as

$$L_t^{\text{joint}}(\mathbf{z}_t, \boldsymbol{\pi}_t, \{\mathbf{a}_{t,k}, \Delta\mathbf{U}_{t,k}\}_{k=1}^K \mid \{\mathbf{U}_{0,k}\}_{k=1}^K) \doteq \frac{1}{2}\left\|\mathbf{z}_t - \sum_{k=1}^K \boldsymbol{\pi}_{t,k}(\mathbf{U}_{0,k} + \Delta\mathbf{U}_{t,k})\mathbf{a}_{t,k}\right\|_2^2. \quad (7)$$

However, under L_t^{joint}, each correction is driven by the combined reconstruction error and thus finding one of the optimal $\Delta\mathbf{U}$ terms depends on all other dictionaries and, likewise, prevents parallel computation. To decouple these terms, note that because $\boldsymbol{\pi}_{t,k} \geq 0$ and $\sum_k \boldsymbol{\pi}_{t,k} = 1$, the convexity of $\|\cdot\|_2^2$ gives the upper bound:

$$\begin{aligned} L_t^{\text{joint}} &\leq \frac{1}{2}\sum_{k=1}^K \boldsymbol{\pi}_{t,k} \|\mathbf{z}_t - (\mathbf{U}_{0,k} + \Delta\mathbf{U}_{t,k})\mathbf{a}_{t,k}\|_2^2 \\ &= \frac{1}{2}\sum_{k=1}^K \boldsymbol{\pi}_{t,k} \|\mathbf{r}_{t,k} - \Delta\mathbf{U}_{t,k}\mathbf{a}_{t,k}\|_2^2, \qquad \mathbf{r}_{t,k} \doteq \mathbf{z}_t - \mathbf{U}_{0,k}\mathbf{a}_{t,k}. \end{aligned} \quad (8)$$

Each correction $\Delta\mathbf{U}_{t,k}$ thus learns to predict the residual $\mathbf{r}_{t,k}$ left by its base dictionary. To prevent overfitting, we also regularize the squared (Frobenius) norm of each correction, $\|\Delta\mathbf{U}_{t,k}\|_F^2$. Together, we arrive at the regularized learning objective for time step t:

$$L_t(\boldsymbol{\pi}_t, \{\mathbf{a}_{t,k}, \mathbf{r}_{t,k}, \Delta\mathbf{U}_{t,k}\}_{k=1}^K) \doteq \frac{1}{2}\sum_{k=1}^K \boldsymbol{\pi}_{t,k} \|\mathbf{r}_{t,k} - \Delta\mathbf{U}_{t,k}\mathbf{a}_{t,k}\|_2^2 + \frac{\lambda}{2}\sum_{k=1}^K \|\Delta\mathbf{U}_{t,k}\|_F^2, \quad (9)$$

where $\lambda \geq 0$. Since L_t separates across dictionaries, each correction can be updated independently.

Unrolled optimization. To derive a recurrent update for each correction, we consider a gradient descent step of size η_k for the objective (9) at the previous $\Delta\mathbf{U}_{t-1,k}$. The resulting update is

$$\Delta\mathbf{U}_{t,k} = (1-\eta_k\lambda)\Delta\mathbf{U}_{t-1,k} + \eta_k\boldsymbol{\pi}_{t,k}\left(\mathbf{r}_{t,k} - (1-\eta_k\lambda)\Delta\mathbf{U}_{t-1,k}\mathbf{a}_{t,k}\right)\mathbf{a}_{t,k}^\top \quad (10)$$

$$= (1-\eta_k\lambda)\Delta\mathbf{U}_{t-1,k}\left(\mathbf{I} - \eta_k\boldsymbol{\pi}_{t,k}\mathbf{a}_{t,k}\mathbf{a}_{t,k}^\top\right) + \eta_k\boldsymbol{\pi}_{t,k}\mathbf{r}_{t,k}\mathbf{a}_{t,k}^\top. \quad (11)$$

The update reduces its component along the current code $\mathbf{a}_{t,k}$, and adds the rank-one term $\mathbf{r}_{t,k}\mathbf{a}_{t,k}^\top$, both scaled by the routing probability $\boldsymbol{\pi}_{t,k}$. The forgetting factor $1-\eta_k\lambda$ decays the contribution of earlier tokens geometrically with t. This gives an online dictionary learning rule that adapts the effective dictionaries to the incoming sequence while retaining the base dictionaries learned offline.

`OnlineTSSA` operator. After updating the fast corrections, we use the effective dictionaries $\mathbf{U}_{t,k} = \mathbf{U}_{0,k} + \Delta\mathbf{U}_{t,k}$ on the encoding side while retaining the base dictionaries $\mathbf{U}_{0,k}$ on the decoding side of the `TSSA` operator defined in (4). This leads to the `OnlineTSSA` operator

$$\texttt{OnlineTSSA}\big(\mathbf{z}_t \mid \{\mathbf{U}_{0,k}, \mathbf{U}_{t,k}\}_{k=1}^K\big) \doteq -\tau\sum_{k=1}^K \boldsymbol{\pi}_{t,k}\mathbf{U}_{0,k}\mathrm{Diag}\left(\nabla f\left[\mathbf{a}_{t,k}^{\odot 2}\right]\right)\mathbf{U}_{t,k}^\top\mathbf{z}_t. \quad (12)$$

Here the sequence statistic in $\mathbf{D}$ is replaced by that of the current code $\mathbf{a}_{t,k}$, so the operator depends only on the current token and the fast corrections. In the gated MLP view of Section 2.1, each branch is thus a gated MLP on the current token whose up projection $\mathbf{U}_{t,k}^\top$ is corrected online, and earlier tokens influence the output only through the corrections $\Delta\mathbf{U}_{t,k}$. This asymmetric use of the dictionaries keeps the stable base dictionaries as decoders while allowing the recurrent corrections to adapt to the current sequence. Figure 1(c) summarizes the resulting operator.

Practical implementation. Due to limited space, we leave implementation details about practical design choices such as parameterization of step sizes to Section C. Here, we highlight three aspects.

1. For the routing, the router for the `OnlineTSSA` module can be made simpler according to (19) given in Section C, and it can be made sparser according to (20): a dictionary with $\boldsymbol{\pi}_{t,k} = 0$ receives no update at token t and does not contribute to the output. Retaining the few largest entries of $\boldsymbol{\pi}_t$ thus restricts the computation to a few dictionaries per token, emulating the role of an MoE operator. Since each branch already has the form of a gated MLP, we can thus replace the standard MLP layer with a sparse `OnlineTSSA` operator, as shown in Figure 1(b).
2. For the decoder, following `TSSA` (Wu et al., 2024), the decoder is untied: each dictionary decodes with a separate matrix $\mathbf{V}_k \in \mathbb{R}^{d\times p}$ learned offline, in place of $\mathbf{U}_{0,k}$ in (12), which unties the down projection of the gated MLP, while $\mathbf{U}_{0,k}$ still produces the codes and the residuals.
3. For hardware-efficient implementation, the recurrence admits parallel training over the sequence. The update (11) is linear in the correction and has the form of the gated delta rule (Yang et al., 2024; 2025a). We therefore adopt the hardware-efficient chunk-wise parallel algorithm developed for DeltaNet, which computes the updates within a chunk of tokens jointly by matrix multiplications and propagates the correction recurrently only from one chunk to the next. We refer readers to Yang et al. (2024) for implementation details of the algorithm.

Offline and online learning. The dictionaries are the only weights that (12) consumes, and the recurrence (10) already corrects them from the tokens alone. A network built from these layers therefore admits two modes of learning, summarized in Algorithm 1. For offline training, the fast corrections start at $\Delta\mathbf{U}_{0,k} = \mathbf{0}$ in every training sequence and are discarded once the sequence ends, so back propagation shapes only the base dictionaries and the remaining weights, and no fast state survives a gradient step. For online learning, the network runs the very same recurrence, so adaptation to the sequence at hand is performed by the forward pass itself, with no labels and no back propagation.

Algorithm 1 Offline and Online Learning for FOLD.

Require: weights θ, including $\{\mathbf{U}_{0,k}\}$; rates η, ρ (online only)
$\Delta\mathbf{U}_{0,k} \leftarrow \mathbf{0}, \forall k$, in every layer
for each sequence $(\mathbf{z}_1, \dots, \mathbf{z}_T)$ **do**
  for each layer **do**
    for $t = 1, \dots, T$ **do**
      update $\Delta\mathbf{U}_{t,k}, \forall k$ ▷ (10)
      $\mathbf{z}_t \leftarrow \mathbf{z}_t + \texttt{OnlineTSSA}(\mathbf{z}_t)$ ▷ (12)
    end for
    apply the `MLP` block, if any
  end for
  if OFFLINE **then**
    $\theta \leftarrow \textsc{Optimizer}(\theta, \nabla_\theta \mathcal{L})$ ▷ offline back prop.
    $\Delta\mathbf{U}_{0,k} \leftarrow \mathbf{0}, \forall k$ ▷ refresh state
  else ONLINE
    $\mathbf{U}_{0,k} \leftarrow \mathbf{U}_{0,k} + \eta\,\Delta\mathbf{U}_{T,k}$, ▷ online fwd. prop. (13)
    $\Delta\mathbf{U}_{0,k} \leftarrow \rho\,\Delta\mathbf{U}_{T,k}, \forall k.$
  end if
end for

The two modes differ at the end of a sequence. After deployment, the final correction $\Delta\mathbf{U}_{T,k}$ is the change to the k-th encoding dictionary accumulated over the sequence. Instead of discarding it, we use it to adjust the base dictionary:

$$\mathbf{U}_{0,k} \leftarrow \mathbf{U}_{0,k} + \eta\,\Delta\mathbf{U}_{T,k}, \qquad \Delta\mathbf{U}_{0,k} \leftarrow \rho\,\Delta\mathbf{U}_{T,k}, \tag{13}$$

where $\eta \in [0,1]$ is the consolidation rate and $\rho \in [0,1]$ is the fraction of the correction carried over to the next sequence. The updated $\mathbf{U}_{0,k}$ is used only in the encoding dictionary $\mathbf{U}_{t,k} = \mathbf{U}_{0,k} + \Delta\mathbf{U}_{t,k}$ of (12); the codes, residuals, and routing weights in (10) are still computed with the pretrained dictionary. With $\eta = \rho = 0$, the correction is discarded after every sequence, as in test-time training. With $\eta = 0$ and $\rho = 1$, it is carried over indefinitely as a recurrent state. With $\eta > 0$ and a small ρ, a fraction η of the correction from each sequence is kept in the base dictionaries, and the next sequence starts from the smaller correction $\rho\,\Delta\mathbf{U}_{T,k}$. Since a network built from `OnlineTSSA` operators can be trained both offline by back propagation and online by forward propagation, we call it a Forward Online Learning Deep (FOLD) network, illustrated in Figure 1(b).

3.2 Connections to Online PCA, DeltaNet, and Kalman Filtering

The recurrence (10) and the operator (12) contain several classical and modern learning rules as special cases. We state these connections here and give the derivations in Section D.

Online PCA. Consider a single dictionary ($K = 1$, so $\boldsymbol{\pi}_{t,1} = 1$) with no base dictionary ($\mathbf{U}_{0,1} = \mathbf{0}$, so $\mathbf{r}_{t,1} = \mathbf{z}_t$ and $\mathbf{U}_{t,1} = \Delta\mathbf{U}_{t,1}$), no regularization ($\lambda = 0$), and codes computed with the current dictionary, $\mathbf{a}_{t,1} = \mathbf{U}_{t-1,1}^\top \mathbf{z}_t$. Writing $\mathbf{U}_t = \mathbf{U}_{t,1}$, the update (10) becomes

$$\mathbf{U}_t = \mathbf{U}_{t-1} + \eta\left(\mathbf{I} - \mathbf{U}_{t-1}\mathbf{U}_{t-1}^\top\right)\mathbf{z}_t\mathbf{z}_t^\top\mathbf{U}_{t-1}, \tag{14}$$

which is Oja's subspace rule for online PCA (Oja, 1982; 1989), whose iterates converge to the principal subspace of the tokens (Yan et al., 1994). The general recurrence extends this rule in three ways: all K dictionaries are updated at every token, with step sizes scaled by the routing weights; each dictionary is a correction to a base dictionary trained offline; and the factor $1 - \eta_k\lambda$ discounts earlier tokens geometrically.

DeltaNet. In Gated DeltaNet (Yang et al., 2025a), a state $\mathbf{S}_t$ that maps keys $\mathbf{k}_t$ to values $\mathbf{v}_t$ is updated by $\mathbf{S}_t = \alpha_t\mathbf{S}_{t-1}(\mathbf{I} - \beta_t\mathbf{k}_t\mathbf{k}_t^\top) + \beta_t\mathbf{v}_t\mathbf{k}_t^\top$, which is a decay by α_t followed by one gradient step of size β_t on $\frac{1}{2}\|\mathbf{v}_t - \mathbf{S}\mathbf{k}_t\|_2^2$; DeltaNet (Schlag et al., 2021; Yang et al., 2024) is the case $\alpha_t = 1$. This is exactly the form of (11) with $\mathbf{S}_t = \Delta\mathbf{U}_{t,k}$, $\mathbf{k}_t = \mathbf{a}_{t,k}$, $\mathbf{v}_t = \mathbf{r}_{t,k}$, $\beta_t = \eta_k\boldsymbol{\pi}_{t,k}$, and $\alpha_t = 1 - \eta_k\lambda$. The difference is in what the state represents. In DeltaNet, keys and values are separate projections of the token, and the state is a matrix, not directly the shape of any parameter, that is discarded at the end of the sequence. In `OnlineTSSA`, the value is the residual of the base dictionary, and the state is a correction to that dictionary, read out through the encoder $(\mathbf{U}_{0,k} + \Delta\mathbf{U}_{t,k})^\top\mathbf{z}_t$. For this reason, the state can be consolidated into the weights by (13).

Kalman filtering. The gate $\mathbf{D}$ in (3) can also be derived from a posterior mean. Suppose $\mathbf{U}_k$ has orthonormal columns, the code of a token in the k-th subspace has the Gaussian prior $\mathcal{N}(\mathbf{0}, \mathrm{Diag}(\mathbf{s}_k))$, where $\mathbf{s}_k = (\mathbf{U}_k^\top\mathbf{Z})^{\odot 2}\boldsymbol{\pi}_k/\langle\boldsymbol{\pi}_k, \mathbf{1}\rangle$ is the vector of token statistics in (3), and the token is observed with isotropic Gaussian noise of variance ϵ^2/d. Then the posterior mean of the code, which is the measurement update of a Kalman filter (Kalman, 1960), is $(\mathbf{I} - (\epsilon^2/d)\mathbf{D}(\mathbf{Z}, \boldsymbol{\pi}_k \mid \mathbf{U}_k))\mathbf{U}_k^\top\mathbf{z}_j$. As a result, with step size $\tau/n = \epsilon^2/d$, the output $\mathbf{z}_j^+$ of the `TSSA` step (4) is the $\boldsymbol{\pi}_{j,k}$-weighted average over k of $\mathbf{z}_j$ with its component in the k-th subspace replaced by the posterior mean $\mathbf{U}_k\mathbb{E}[\mathbf{a} \mid \mathbf{z}_j]$. In `OnlineTSSA`, the encoding dictionaries, which play the role of the measurement matrix in this model, are updated at every token by (10).

4 Experimental Verification

In this section, we compare the proposed FOLD architecture with baselines. Section 4.1 studies the offline learning setting, where all models are trained with backpropagation, and shows that FOLD performs competitively. Section 4.2 further considers the online learning setting, where we demonstrate the ability of FOLD to learn continuously via forward propagation.

4.1 Learning via Backpropagation for Short- and Long-Context Tasks

We compare FOLD models with baselines including TTT and Transformer models; all models are pre-trained with backpropagation alone. These models are evaluated on both short- and long-context tasks. The main message is that FOLD has the same complexity as TTT models, while it achieves comparable or superior performance to the state-of-the-art TTT models.

Baselines. Following prior work (Gu & Dao, 2023; Yang et al., 2025a), we include Transformer++ as a self-attention-based architecture. Further, we consider TTT models including DeltaNet (Yang et al., 2024), Gated DeltaNet (Yang et al., 2025a), and a variant FOLD-MLP of FOLD that adopts MLP in place of sparse `OnlineTSSA` operators. These models share the same network structure (including MLP) and differ only in their TTT operator. Finally, we include the proposed FOLD network, which is also a TTT model, with the main difference that it adopts both a dense and a sparse `OnlineTSSA` operator. More details are provided in Section E.1.

Training. We train the models on 15B tokens of SlimPajama (Soboleva et al., 2023) with sequences of length 4,096 and a global batch size of 512 sequences. The weight matrices inside the blocks are optimized with Muon (Jordan et al., 2024) (momentum 0.95, Nesterov, 5 Newton–Schulz iterations) and the embeddings, gains, and biases with AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.95$). Both optimizers use a peak learning rate of 10^{-3}, weight decay 0.1, linear warmup over the first 1% of the tokens, cosine decay to 10^{-4}, and gradient clipping at 1.0, in `bfloat16` mixed precision.

Short-context tasks. To evaluate the language modeling ability of various models, we consider commonsense reasoning tasks from the LM-harness (Gao et al., 2024): PIQA, HellaSwag, WinoGrande, ARC-Easy, ARC-Challenge, BoolQ, SciQ, COPA, and OpenBookQA. The evaluation is conducted zero-shot on the final checkpoints of the trained models. Table 1 reports the performance of the models on these tasks. As observed, all models yield comparable average performance.

Table 1: Zero-shot accuracy (%) of compared models on the commonsense reasoning tasks from LM-harness. Bold marks the best value in each column; higher is better.

Model	PIQA acc ↑	HellaSwag acc_n ↑	WinoGrande acc ↑	ARC-E acc ↑	ARC-C acc_n ↑	BoolQ acc ↑	SciQ acc ↑	COPA acc ↑	OBQA acc_n ↑	Avg
Transformer++	65.0	36.4	51.2	45.0	23.6	**61.6**	77.5	65.0	29.8	50.6
DeltaNet	64.5	36.1	51.3	43.9	**24.6**	59.9	78.3	66.0	26.6	50.1
Gated DeltaNet	**66.7**	**37.6**	**52.4**	46.3	23.4	58.4	77.8	64.0	**30.0**	50.7
FOLD (Ours)	64.8	35.7	49.6	45.6	24.1	58.0	**80.1**	69.0	**30.0**	50.8
FOLD-MLP (Ours)	66.4	36.5	50.4	**47.9**	23.2	59.7	78.5	**70.0**	29.6	**51.4**

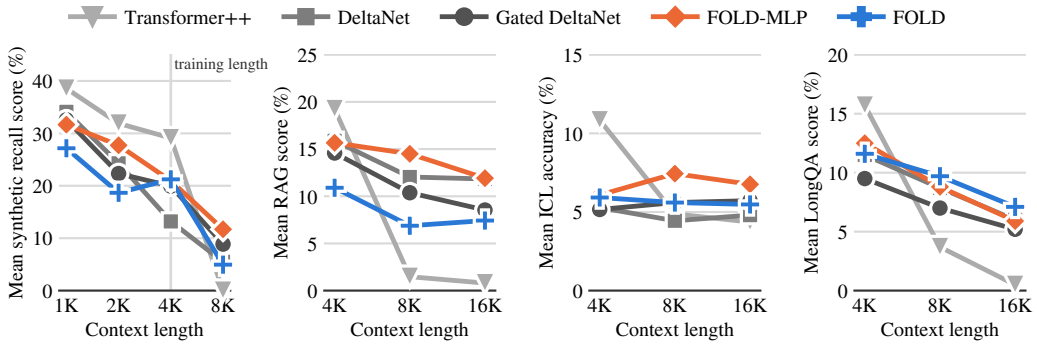


Figure 2: Mean score (%) against context length on the four categories of long-context tasks: synthetic recall, retrieval-augmented generation (RAG), many-shot in-context learning (ICL) and long-document question answering (LongQA). All models are trained at length 4K.

Long-context tasks. We follow prior work (Yang et al., 2025a; Hatamizadeh et al., 2026; Peng et al., 2026) and consider four categories of long-context tasks: synthetic recall, retrieval-augmented generation (RAG), many-shot in-context learning (ICL), and long-document question answering (LongQA). Synthetic recall and LongQA use tasks from RULER (Hsieh et al., 2024), and RAG and ICL use tasks from HELMET (Yen et al., 2025), at context lengths of up to four times the training length. More details can be found in Section E.1. Figure 2 shows the mean score of each category against the context length. As seen, Transformer++ outperforms all TTT models when the context length at test time does not exceed that at training time, while it suffers from a sharp performance drop when the former exceeds the latter. Meanwhile, the performance of TTT models degrades more gracefully as the test context length increases. Among the TTT models, FOLD-MLP is particularly competitive, while FOLD appears slightly worse.

4.2 Forward Learning on Continuous Associative Recall Tasks

We next test whether a pretrained network can keep learning new associations by forward propagation alone, without forgetting the ones it has already learned.

Setup. Multi-query associative recall (MQAR) (Arora et al., 2024) evaluates the ability of a model to recall information within a context when it is faced with multiple recall queries: the context presents key–value pairs and then queries their keys. Following the same work, each network has two blocks of width 256, a convolution layer followed by a recurrent operator, which is either `OnlineTSSA` or Gated DeltaNet. We pretrain each network on MQAR by back propagation until it recalls 99% of 512 key–value pairs presented in one context. Forward learning then runs the online mode of Algorithm 1 on the FOLD network, with $\eta = 0.05$, selected by a grid search, and $\rho = 0$. We compare it with both networks without consolidation ($\eta = 0$ in (13)) in two forms. In the first, the recurrent state is cleared at every context ($\rho = 0$). In the second, the state is carried across the whole stream ($\rho = 1$), so that it is the only memory of the previously visited key-value pairs. Gated DeltaNet has no counterpart of forward learning, since its recurrent state is not a correction to its weights.

Task. Each network reads a stream of 32 groups of 32 new key–value pairs, 1,024 pairs in total and twice the largest load seen in pretraining (Figure 3). Each group is presented in eight consecutive contexts and never appears again, and each context writes the pairs of its group and then queries their keys. We measure learning by the accuracy on a group in the first context in which it appears,

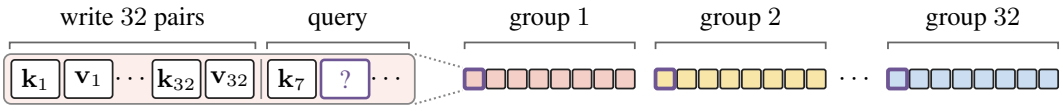


Figure 3: The stream of associations. Each context writes the 32 key–value pairs of its group and then queries their keys (left: the first context of group 1). Each group is presented in eight consecutive contexts and never appears again. Learning is measured in the first context of each group (outlined), and retention by querying groups 1 to 28 at the end of the stream.

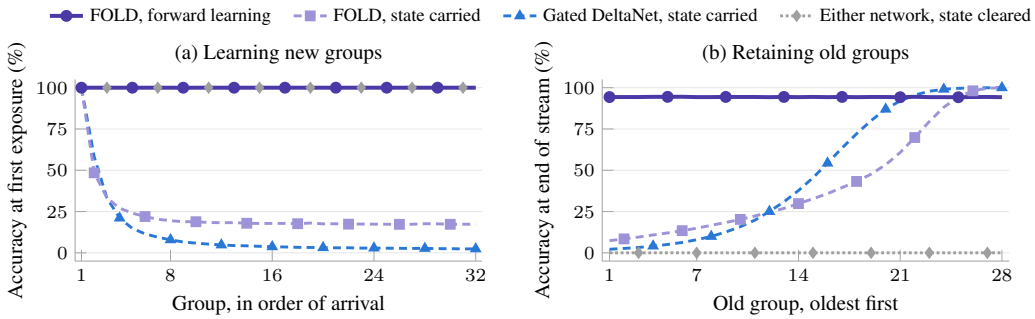


Figure 4: Online learning on the stream of 32 groups. (a) Accuracy on each group in the first context in which it appears. (b) Accuracy on each old group at the end of the stream.

and retention by the accuracy on the old groups, that is, all groups except the four most recent, at the end of the stream.

Results. Figure 4 shows the results. Without consolidation, each network loses one of the two abilities. With the state cleared, every context starts from the pretrained weights, so the network learns each group perfectly in its first context but retains nothing after the group has passed. With the state carried, the network retains part of the stream, but the accumulated state interferes with new writes. After the first few groups, the accuracy on a new group in its first context falls to about 17% for FOLD and below 3% for Gated DeltaNet, and three to four contexts are needed to learn the group. The carried state also retains only the most recent groups: both networks recall the last few groups almost perfectly and have lost nearly all of the oldest ones (Figure 4(b)).

Yet, forward learning keeps both abilities. At the end of each context, consolidation moves what the context has taught from the recurrent state into the weights and clears the state, so that each new context starts from an empty state, as with the state cleared, while information about the old groups remains in the weights. Forward learning thus learns every group in its first context, and at the end of the stream it recalls 94.4% of the old pairs, against 48.9% for Gated DeltaNet and 42.1% for FOLD with the state carried. No labels or gradients are used at any point of the stream, and the pairs are written into the weights of the FOLD network by the forward pass alone.

5 Conclusion

In this work, we proposed the Forward Online Learning Deep (FOLD) architecture based on the novel linear-complexity `OnlineTSSA` operator, which unifies both sequence-mixing (attention) and feature-mixing (MLP/MoE) components of Transformers by a common type of operators. FOLD's novel and principled construction from the perspective of online corrections to learned dictionary bases enables training both via standard offline back propagation as well as supervision-free explicit forward propagation. FOLD may then seamlessly incorporate new knowledge or distribution shifts during inference while avoiding catastrophic forgetting. Our empirical evaluation validates that FOLD achieves competitive performance with SOTA methods across numerous short- and long-context language benchmarks and superior performance on continuous associative recall learning using just forward online consolidation, without relying solely on back propagation. Compared to the common practice with back propagation, online learning via forward propagation has clear advantages in minimal data requirement, computational efficiency, and no need of supervision. Note that the proposed forward online learning mechanism is still rather primitive and is subject to significant improvements in the future, opening the door to significantly more data-efficient and power-efficient online learning algorithms.

AI USAGE STATEMENT

We used generative AI tools for assistance in this work. AI tools were utilized to develop and write theoretical derivations, all of which the authors reviewed, revised, and edited. They also helped implement and execute the experiments, all of which were designed and verified by multiple authors. In addition, critical aspects of the implementation were manually written and thoroughly tested. Finally, AI tools were used to help draft and edit the tables and figures, in the appendices and in parts of the main text, under the supervision of the authors. The authors take full responsibility for the content of this paper, including all theoretical claims, derivations, experimental results, figures, and text produced with AI assistance.

REFERENCES

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Y Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. In *International conference on learning representations*, volume 2024, pp. 15664–15730, 2024.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

Kwan Ho Ryan Chan, Yaodong Yu, Chong You, Haozhi Qi, John Wright, and Yi Ma. Redunet: A white-box deep network from the principle of maximizing rate reduction. *Journal of Machine Learning Research*, 23(114):1–103, 2022. URL http://jmlr.org/papers/v23/21-0631.html.

Tsung-Han Chan, Kui Jia, Shenghua Gao, Jiwen Lu, Zinan Zeng, and Yi Ma. Pcanet: A simple deep learning baseline for image classification? *IEEE Transactions on Image Processing*, 24(12): 5017–5032, 2015. doi: 10.1109/TIP.2015.2475625.

Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. Sparse autoencoders find highly interpretable features in language models. *arXiv preprint arXiv:2309.08600*, 2023.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, July 2024. URL https://zenodo.org/records/12608602.

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

Ali Hatamizadeh, Yejin Choi, and Jan Kautz. Gated DeltaNet-2: Decoupling erase and write in linear attention. *arXiv preprint arXiv:2605.22791*, 2026. URL https://arxiv.org/abs/2605.22791.

Geoffrey Hinton. The forward-forward algorithm: Some preliminary investigations. *arXiv preprint arXiv:2212.13345*, 2022.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Rudolph E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

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.

Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

Erkki Oja. Neural networks, principal components, and subspaces. *International Journal of Neural Systems*, 1(1):61–68, 1989.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Leon Derczynski, et al. Rwkv: Reinventing rnns for the transformer era. In *Findings of the association for computational linguistics: EMNLP 2023*, pp. 14048–14077, 2023a.

Liangzu Peng, Paris Giampouras, and Rene Vidal. The ideal continual learner: An agent that never forgets. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 27585–27610. PMLR, 23–29 Jul 2023b. URL https://proceedings.mlr.press/v202/peng23a.html.

Liangzu Peng, Aditya Chattopadhyay, Luca Zancato, Elvis Nunez, Wei Xia, and Stefano Soatto. Gated kalmanet: A fading memory layer through test-time ridge regression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 20212–20222, 2026.

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

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627b token cleaned and deduplicated version of RedPajama, June 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

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.

Shengbang Tong, Xili Dai, Ziyang Wu, Mingyang Li, Brent Yi, and Yi Ma. Incremental learning of structured memory via closed-loop transcription. *arXiv preprint arXiv:2202.05411*, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Rene Vidal. Attention: Self-Expression Is All You Need. October 2021. URL https://openreview.net/pdf?id=MmujBClawFo.

Ziyang Wu, Christina Baek, Chong You, and Yi Ma. Incremental learning via rate reduction. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1125–1133. IEEE, 2021.

Ziyang Wu, Tianjiao Ding, Yifu Lu, Druv Pai, Jingyuan Zhang, Weida Wang, Yaodong Yu, Yi Ma, and Benjamin D Haeffele. Token statistics transformer: Linear-time attention via variational rate reduction. *arXiv preprint arXiv:2412.17810*, 2024.

Wei-Yong Yan, Uwe Helmke, and John B. Moore. Global analysis of Oja's flow for neural networks. *IEEE Transactions on Neural Networks*, 5(5):674–683, 1994.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems*, volume 37, pp. 115491–115522, 2024. doi: 10.52202/079017-3668.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=r8H7xhYPwz.

Zitong Yang, Neil Band, Shuangping Li, Emmanuel Candes, and Tatsunori Hashimoto. Synthetic continued pretraining. In *International Conference on Learning Representations*, volume 2025, pp. 44379–44421, 2025b.

Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. HELMET: How to evaluate long-context language models effectively and thoroughly. In *International Conference on Learning Representations*, 2025.

Yaodong Yu, Kwan Ho Ryan Chan, Chong You, Chaobing Song, and Yi Ma. Learning diverse and discriminative representations via the principle of maximal coding rate reduction. *Advances in neural information processing systems*, 33:9422–9434, 2020.

Yaodong Yu, Sam Buchanan, Druv Pai, Tianzhe Chu, Ziyang Wu, Shengbang Tong, Benjamin D Haeffele, and Yi Ma. White-box transformers via sparse rate reduction. *arXiv preprint arXiv:2306.01129*, 2023.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. TinyLlama: An open-source small language model, 2024.

A Variational Spectral Bound

Section 2.1 replaces the $\log\det$ in the compression term $R_c(\mathbf{Z}, \mathbf{\Pi})$ by a sum of scalar functions of diagonal entries. The substitution is justified by the following bound, proved in Wu et al. (2024).

Theorem 1 (Variational spectral bound): *Let $f\colon [0,\infty) \to \mathbb{R}$ be non-decreasing, concave, and obey $f(0) = 0$, and let $F\colon \mathrm{PSD}(d) \to \mathbb{R}$ have the form $F(\mathbf{M}) = \sum_{i=1}^{d} f(\lambda_i(\mathbf{M}))$. Then for each $\mathbf{M} \in \mathrm{PSD}(d)$ and $\mathbf{Q} \in \mathrm{O}(d)$, we have*

$$F(\mathbf{M}) \le \sum_{i=1}^{d} f\left((\mathbf{Q}^\top \mathbf{M} \mathbf{Q})_{ii}\right). \tag{15}$$

Further, the inequality in (15) is achieved with equality for any $\mathbf{Q}$ which diagonalizes $\mathbf{M}$, and if f is strictly concave then the inequality in (15) is achieved with equality if and only if $\mathbf{Q}$ diagonalizes $\mathbf{M}$.

B Connection to the Original Compression Term

This appendix relates the variational operator of Section 2.1 to a gradient step on the original compression term $R_c(\mathbf{Z}, \mathbf{\Pi})$ in (2). The connection is useful for interpreting the diagonal gate $\mathbf{D}$, but is not required for the development in Section 3.1.

To connect TSSA to the original compression term in (2), consider its gradient with respect to $\mathbf{Z}$:

$$\nabla_{\mathbf{Z}} R_c(\mathbf{Z}, \mathbf{\Pi}) = \alpha \sum_{k=1}^{K} \left(\mathbf{I} + \alpha_k \mathbf{Z}\mathrm{Diag}(\boldsymbol{\pi}_k)\mathbf{Z}^\top\right)^{-1} \mathbf{Z}\mathrm{Diag}(\boldsymbol{\pi}_k), \tag{16}$$

where $\alpha_k = d/(\epsilon^2 n_k)$ and $\alpha = d/(\epsilon^2 n)$, as defined in Section 2.1.

Suppose $\mathrm{rank}(\mathbf{Z}\mathrm{Diag}(\boldsymbol{\pi}_k)\mathbf{Z}^\top) = p$ and

$$\mathbf{Z}\mathrm{Diag}(\boldsymbol{\pi}_k)\mathbf{Z}^\top = \mathbf{U}_k \mathbf{\Lambda}_k \mathbf{U}_k^\top, \qquad \mathbf{\Lambda}_k = \mathrm{Diag}(\lambda_1, \ldots, \lambda_p),$$

with $\mathbf{U}_k \in \mathrm{O}(d, p)$. A gradient step for token $\mathbf{z}_j$ then becomes

$$\mathbf{z}_j - \tau \nabla_{\mathbf{Z}} R_c(\mathbf{Z}, \mathbf{\Pi}) = \mathbf{z}_j - \tau\alpha \sum_{k=1}^{K} \boldsymbol{\pi}_{j,k} \left(\mathbf{I} + \alpha_k \mathbf{Z}\mathrm{Diag}(\boldsymbol{\pi}_k)\mathbf{Z}^\top\right)^{-1} \mathbf{z}_j \tag{17}$$

$$= \mathbf{z}_j - \tau\alpha \sum_{k=1}^{K} \boldsymbol{\pi}_{j,k} \mathbf{U}_k g(\mathbf{\Lambda}_k) \mathbf{U}_k^\top \mathbf{z}_j, \tag{18}$$

where

$$g(\mathbf{\Lambda}_k) = \mathrm{Diag}\left(\frac{1}{1+\alpha_k\lambda_1}, \ldots, \frac{1}{1+\alpha_k\lambda_p}\right).$$

Thus, the diagonal operator $\mathbf{D}$ in (3) approximates the spectral filter $g(\mathbf{\Lambda}_k)$ and agrees with it when $\mathbf{U}_k$ diagonalizes $\mathbf{Z}\mathrm{Diag}(\boldsymbol{\pi}_k)\mathbf{Z}^\top$. Computing this eigen-decomposition at every layer and iteration would recover the spectral quantities directly, but would be substantially more expensive and may introduce numerical instability.

C Practical Implementation of OnlineTSSA

This appendix details the design choices summarized in the practical implementation paragraph of Section 3.1.

Codes. We further promote sparsity of the codes $\mathbf{a}_{t,k}$. Sparse autoencoders achieve this with a ReLU encoder (Cunningham et al., 2023). Here we adopt its smooth counterpart, SiLU, and compute the code as $\mathbf{a}_{t,k} \doteq \mathrm{SiLU}(\mathbf{U}_{0,k}^\top \mathbf{z}_t)$. For the code that enters the recurrent update, we additionally apply a causal short convolution to $\mathbf{U}_{0,k}^\top \mathbf{z}_t$ before the SiLU, as in DeltaNet and related linear models (Yang et al., 2024; 2025a), so that each update sees a short window of preceding tokens.

Routing. In `TSSA`, the weights (5) score a token by the energy $\|\mathbf{U}_k^\top \mathbf{z}_t\|_2^2$ of its code in each subspace. For the (dense) `OnlineTSSA`, we use a simpler score that is also computed from the base dictionaries, so the router has no additional parameters:

$$\boldsymbol{\pi}_{t,k} \doteq \frac{\sigma(c_{t,k}/\gamma)}{\sum_{j=1}^{K} \sigma(c_{t,j}/\gamma)}, \qquad c_{t,k} \doteq \cos(\mathbf{z}_t, \mathbf{U}_{0,k}\mathbf{1}) = \frac{\mathbf{1}^\top \mathbf{U}_{0,k}^\top \mathbf{z}_t}{\|\mathbf{z}_t\|_2 \|\mathbf{U}_{0,k}\mathbf{1}\|_2}, \tag{19}$$

where $\mathbf{1} \in \mathbb{R}^p$ is the all-ones vector, σ is the sigmoid function, and $\gamma > 0$ is a temperature. The score $c_{t,k}$ is the sum of the entries of the code $\mathbf{U}_{0,k}^\top \mathbf{z}_t$, normalized by the norms of the token and of the sum of the atoms. It thus replaces the sum of the squared entries in (5) by the sum of the entries, and it lies in $[-1, 1]$ for any scale of the token or the atoms. The weights are the sigmoids of the scores, normalized to sum to one. When $c_{t,k}/\gamma$ is well below zero, $\sigma(c_{t,k}/\gamma) \approx e^{c_{t,k}/\gamma}$, so (19) is close to a softmax of the scores. When it is well above zero, $\sigma(c_{t,k}/\gamma) \approx 1$, so several dictionaries that are well aligned with the token receive nearly equal weights, whereas a softmax would put most of the weight on the largest score. This router performs on par with a separately learned linear softmax router (Table 4).

For the routing of the sparse `OnlineTSSA`, let $\mathcal{K}_t \subset [K]$ index the $s < K$ largest weights of the token $\mathbf{z}_t$. The weights are replaced by

$$\tilde{\boldsymbol{\pi}}_{t,k} \doteq \begin{cases} \boldsymbol{\pi}_{t,k} / \sum_{j \in \mathcal{K}_t} \boldsymbol{\pi}_{t,j}, & k \in \mathcal{K}_t, \\ 0, & \text{otherwise}, \end{cases} \tag{20}$$

so that at each token only s corrections receive a rank-one update in (10) and only s dictionaries contribute to (12). With three of nine dictionaries active, the loss increases by about 0.05 relative to dense routing (Table 4).

Step size and decay. In (10), the step size η_k and the decay $1 - \eta_k\lambda$ are constants. We instead let the step size depend on the current token, $\eta_{t,k} \doteq \sigma(\mathbf{w}_k^\top \mathbf{z}_t) \in (0, 1)$ with a learned $\mathbf{w}_k \in \mathbb{R}^d$, and set the decay to $\alpha_{t,k} \doteq e^{-\lambda_k \eta_{t,k}}$ with a learned $\lambda_k > 0$. This choice follows the derivation: since $e^{-x} \approx 1 - x$ for small x, $\alpha_{t,k} \approx 1 - \eta_{t,k}\lambda_k$, which is the decay in (10) with step size $\eta_{t,k}$. Finally, we divide the step size by the squared code norm. Written in the form of (11), the update becomes

$$\Delta\mathbf{U}_{t,k} = \alpha_{t,k}\Delta\mathbf{U}_{t-1,k}\left(\mathbf{I} - \frac{\eta_{t,k}\boldsymbol{\pi}_{t,k}}{\|\mathbf{a}_{t,k}\|_2^2}\mathbf{a}_{t,k}\mathbf{a}_{t,k}^\top\right) + \frac{\eta_{t,k}\boldsymbol{\pi}_{t,k}}{\|\mathbf{a}_{t,k}\|_2^2}\mathbf{r}_{t,k}\mathbf{a}_{t,k}^\top. \tag{21}$$

The state transition $\alpha_{t,k}\big(\mathbf{I} - \eta_{t,k}\boldsymbol{\pi}_{t,k}\mathbf{a}_{t,k}\mathbf{a}_{t,k}^\top/\|\mathbf{a}_{t,k}\|_2^2\big)$ then has eigenvalues bounded in $(0, 1)$ regardless of the code norm, which keeps the recurrent update stable.

Untied decoder. The decoder $\mathbf{V}_k$ replaces $\mathbf{U}_{0,k}$ only on the decoding side of (12); the codes, the residuals, and the router are still computed with $\mathbf{U}_{0,k}$ (Figure 1(c)). Tying $\mathbf{V}_k = \mathbf{U}_{0,k}$ reduces the parameter count at a small cost in language-modeling loss (Section E.1). Before decoding, each per-dictionary output is processed by an RMSNorm layer (Zhang & Sennrich, 2019) for improved training stability.

Parallel training. Evaluating (21) one token at a time is sequential and therefore slow to train on GPUs. The update, however, has the same form as the DeltaNet recurrence (Yang et al., 2024) with a scalar decay (Yang et al., 2025a): at each step, the state is scaled by $\alpha_{t,k}$ and corrected by a rank-one term formed from the prediction error $\mathbf{r}_{t,k} - \alpha_{t,k}\Delta\mathbf{U}_{t-1,k}\mathbf{a}_{t,k}$ and the code $\mathbf{a}_{t,k}$. The chunk-wise parallel algorithm developed for DeltaNet splits the sequence into chunks, computes all updates within a chunk jointly with matrix multiplications, and propagates the state recurrently only from one chunk to the next. We refer to Yang et al. (2024; 2025a) for details.

D Connections to Online PCA, DeltaNet, and Kalman Filtering

This appendix gives the derivations behind Section 3.2.

Online PCA. In online PCA, the principal p-dimensional subspace of a stream of tokens $\mathbf{z}_1, \mathbf{z}_2, \ldots \in \mathbb{R}^d$ is estimated by minimizing the reconstruction error $\frac{1}{2}\|\mathbf{z}_t - \mathbf{U}\mathbf{U}^\top \mathbf{z}_t\|_2^2$ over $\mathbf{U} \in \mathbb{R}^{d \times p}$ one token at a time. Oja's subspace rule (Oja, 1989) is a gradient step of size η on $\frac{1}{2}\|\mathbf{z}_t - \mathbf{U}\mathbf{a}_t\|_2^2$ with the code $\mathbf{a}_t = \mathbf{U}_{t-1}^\top \mathbf{z}_t$ held fixed, whose gradient in $\mathbf{U}$ is $-(\mathbf{z}_t - \mathbf{U}\mathbf{a}_t)\mathbf{a}_t^\top$:

$$\mathbf{U}_t = \mathbf{U}_{t-1} + \eta\,(\mathbf{z}_t - \mathbf{U}_{t-1}\mathbf{a}_t)\,\mathbf{a}_t^\top = \mathbf{U}_{t-1} + \eta\,\big(\mathbf{I} - \mathbf{U}_{t-1}\mathbf{U}_{t-1}^\top\big)\,\mathbf{z}_t\mathbf{z}_t^\top\mathbf{U}_{t-1}. \tag{22}$$

For $p = 1$ this is Oja's rule $\mathbf{u}_t = \mathbf{u}_{t-1} + \eta\, y_t(\mathbf{z}_t - y_t\mathbf{u}_{t-1})$ with $y_t = \mathbf{u}_{t-1}^\top \mathbf{z}_t$ (Oja, 1982). If the tokens are i.i.d. with covariance $\mathbf{\Sigma}$ and the step sizes satisfy the usual stochastic approximation conditions, the iterates of (22) follow the flow $\dot{\mathbf{U}} = (\mathbf{I} - \mathbf{U}\mathbf{U}^\top)\mathbf{\Sigma}\mathbf{U}$, which, when the p-th and $(p+1)$-th eigenvalues of $\mathbf{\Sigma}$ differ, converges to an orthonormal basis of the span of the top p eigenvectors of $\mathbf{\Sigma}$ for almost every initialization with full column rank (Yan et al., 1994).

Now set $K = 1$, $\mathbf{U}_{0,1} = \mathbf{0}$, and $\lambda = 0$ in (10). Then $\boldsymbol{\pi}_{t,1} = 1$, $\mathbf{r}_{t,1} = \mathbf{z}_t$, and $\mathbf{U}_{t,1} = \Delta\mathbf{U}_{t,1}$, and the update reads $\mathbf{U}_{t,1} = \mathbf{U}_{t-1,1} + \eta_1(\mathbf{z}_t - \mathbf{U}_{t-1,1}\mathbf{a}_{t,1})\mathbf{a}_{t,1}^\top$. With the code computed from the current dictionary, $\mathbf{a}_{t,1} = \mathbf{U}_{t-1,1}^\top \mathbf{z}_t$, this is (22). In `OnlineTSSA`, the code is instead computed from the base dictionary, $\mathbf{a}_{t,k} = \mathbf{U}_{0,k}^\top \mathbf{z}_t$, so $\Delta\mathbf{U}_{t,k}$ is fit to reconstruct the residual $\mathbf{r}_{t,k}$ from a fixed encoding. The recurrence is then linear in the state, which is needed for the chunk-wise parallel training of Section C.

DeltaNet. In linear attention (Katharopoulos et al., 2020), the output at time t is computed from queries $\mathbf{q}_t$, keys $\mathbf{k}_t$, and values $\mathbf{v}_t$, all linear projections of the token $\mathbf{z}_t$, as

$$\mathbf{o}_t = \Big(\sum_{s=1}^{t} \mathbf{v}_s\mathbf{k}_s^\top\Big)\mathbf{q}_t = \mathbf{S}_t\mathbf{q}_t, \qquad \mathbf{S}_t = \mathbf{S}_{t-1} + \mathbf{v}_t\mathbf{k}_t^\top. \tag{23}$$

Every key–value pair is added with the same weight, so old associations are never overwritten, and the norm of $\mathbf{S}_t$ can grow without bound. DeltaNet (Schlag et al., 2021; Yang et al., 2024) uses the delta rule instead, a gradient step of size $\beta_t \in (0,1)$ on the regression loss $\frac{1}{2}\|\mathbf{v}_t - \mathbf{S}\mathbf{k}_t\|_2^2$ at $\mathbf{S} = \mathbf{S}_{t-1}$:

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \beta_t\,(\mathbf{v}_t - \mathbf{S}_{t-1}\mathbf{k}_t)\,\mathbf{k}_t^\top = \mathbf{S}_{t-1}\left(\mathbf{I} - \beta_t\mathbf{k}_t\mathbf{k}_t^\top\right) + \beta_t\mathbf{v}_t\mathbf{k}_t^\top. \tag{24}$$

In Gated DeltaNet (Yang et al., 2025a), the state is first multiplied by a decay $\alpha_t \in (0,1)$, and the same step is then taken at $\alpha_t\mathbf{S}_{t-1}$:

$$\mathbf{S}_t = \alpha_t\mathbf{S}_{t-1}\left(\mathbf{I} - \beta_t\mathbf{k}_t\mathbf{k}_t^\top\right) + \beta_t\mathbf{v}_t\mathbf{k}_t^\top. \tag{25}$$

The update (10) is derived in the same way, with the decay coming from the weight penalty in (9), and its collected form (11) is (25) with

$$\mathbf{S}_t = \Delta\mathbf{U}_{t,k}, \quad \mathbf{k}_t = \mathbf{a}_{t,k}, \quad \mathbf{v}_t = \mathbf{r}_{t,k}, \quad \beta_t = \eta_k\boldsymbol{\pi}_{t,k}, \quad \alpha_t = 1 - \eta_k\lambda.$$

The normalization by $\|\mathbf{a}_{t,k}\|_2^2$ in (21) corresponds to the ℓ^2 normalization of the keys in DeltaNet (Yang et al., 2024). There are three differences. First, the key is the code of the token under the base dictionary, and the value is the part of the token that this code does not reconstruct, so keys and values are tied to the same dictionary. Second, all K states are updated at every token, with step sizes scaled by the routing weights. Third, the output of DeltaNet is $\mathbf{S}_t\mathbf{q}_t$, while in `OnlineTSSA` the state enters the output through its transpose, $\Delta\mathbf{U}_{t,k}^\top\mathbf{z}_t$, as part of the encoder $\mathbf{U}_{t,k}^\top\mathbf{z}_t$ in (12). Since the state is a correction to the encoding dictionary, it can be added to the weights, as in (13). DeltaNet has no weight matrix to which its state could be added.

Kalman filtering. Fix a subspace k, and let $\mathbf{U}_k \in \mathbb{R}^{d\times p}$ have orthonormal columns. Consider the linear Gaussian model

$$\mathbf{z} = \mathbf{U}_k\mathbf{a} + \boldsymbol{\varepsilon}, \qquad \mathbf{a} \sim \mathcal{N}(\mathbf{0}, \mathrm{Diag}(\mathbf{s})), \qquad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma_0^2\mathbf{I}), \tag{26}$$

with $\mathbf{s} \in \mathbb{R}_+^p$ and $\sigma_0^2 = \epsilon^2/d$. The posterior mean of the code is the measurement update of a Kalman filter (Kalman, 1960) with zero prior mean, prior covariance $\mathrm{Diag}(\mathbf{s})$, and measurement matrix $\mathbf{U}_k$:

$$\mathbb{E}[\mathbf{a} \mid \mathbf{z}] = \mathrm{Diag}(\mathbf{s})\mathbf{U}_k^\top\left(\mathbf{U}_k\mathrm{Diag}(\mathbf{s})\mathbf{U}_k^\top + \sigma_0^2\mathbf{I}\right)^{-1}\mathbf{z} = \left(\mathrm{Diag}(\mathbf{s}) + \sigma_0^2\mathbf{I}\right)^{-1}\mathrm{Diag}(\mathbf{s})\mathbf{U}_k^\top\mathbf{z},$$

where the second equality follows from $\mathbf{U}_k^\top(\mathbf{U}_k\mathrm{Diag}(\mathbf{s})\mathbf{U}_k^\top + \sigma_0^2\mathbf{I}) = (\mathrm{Diag}(\mathbf{s}) + \sigma_0^2\mathbf{I})\mathbf{U}_k^\top$, which uses $\mathbf{U}_k^\top\mathbf{U}_k = \mathbf{I}$. For $f(x) = \log(1 + x/\sigma_0^2)$ we have $\nabla f(x) = 1/(\sigma_0^2 + x)$, so the i-th diagonal entry of the gain is $s_i/(\sigma_0^2 + s_i) = 1 - \sigma_0^2\nabla f(s_i)$, and

$$\mathbb{E}[\mathbf{a} \mid \mathbf{z}] = \left(\mathbf{I} - \sigma_0^2\mathrm{Diag}(\nabla f[\mathbf{s}])\right)\mathbf{U}_k^\top\mathbf{z}. \tag{27}$$

Take $\mathbf{s} = \mathbf{s}_k \doteq (\mathbf{U}_k^\top\mathbf{Z})^{\odot 2}\boldsymbol{\pi}_k/\langle\boldsymbol{\pi}_k, \mathbf{1}\rangle$, the second moments of the codes of the tokens assigned to subspace k. Then $\mathrm{Diag}(\nabla f[\mathbf{s}_k]) = \mathbf{D}(\mathbf{Z}, \boldsymbol{\pi}_k \mid \mathbf{U}_k)$ as defined in (3). Decomposing $\mathbf{z}_j = \mathbf{U}_k\mathbf{U}_k^\top\mathbf{z}_j + (\mathbf{I} - \mathbf{U}_k\mathbf{U}_k^\top)\mathbf{z}_j$ and using (27),

$$\mathbf{z}_j - \sigma_0^2\,\mathbf{U}_k\mathbf{D}(\mathbf{Z}, \boldsymbol{\pi}_k \mid \mathbf{U}_k)\mathbf{U}_k^\top\mathbf{z}_j = \mathbf{U}_k\,\mathbb{E}[\mathbf{a} \mid \mathbf{z} = \mathbf{z}_j] + \left(\mathbf{I} - \mathbf{U}_k\mathbf{U}_k^\top\right)\mathbf{z}_j.$$

Since $\sum_k \boldsymbol{\pi}_{j,k} = 1$, the TSSA step (4) with $\tau/n = \sigma_0^2$ is the average of these per-subspace updates with the weights $\boldsymbol{\pi}_{j,k}$:

$$\mathbf{z}_j^+ = \sum_{k=1}^{K} \boldsymbol{\pi}_{j,k} \left(\mathbf{U}_k \, \mathbb{E}[\mathbf{a} \mid \mathbf{z} = \mathbf{z}_j] + \left(\mathbf{I} - \mathbf{U}_k \mathbf{U}_k^\top \right) \mathbf{z}_j \right),$$

where the posterior in the k-th term is taken under (26) with $\mathbf{s} = \mathbf{s}_k$. That is, in the k-th subspace, the i-th code coordinate is multiplied by $s_{k,i}/(\sigma_0^2 + s_{k,i})$, so the coordinates with small variance among the tokens of the subspace are shrunk the most. This is consistent with the derivation of TSSA as a compression step on the tokens of each subspace (Section 2.1). In OnlineTSSA, the prior variances are replaced by the squared codes of the current token, and the encoding dictionary, which plays the role of the measurement matrix, is updated at every token by (10). Since the encoding and decoding dictionaries differ, the identity above does not hold exactly for OnlineTSSA.

E Experiment Details

E.1 Additional Details for Section 4.1

Baselines. All architectures adopt a Triton implementation and apply a causal short convolution of width 4. The backbone follows the 0.4B configuration from Yang et al. (2025a): 11 pre-norm blocks of hidden width 1,536, each a sequence-mixing layer followed by a SwiGLU MLP of intermediate width 6,144, with RMS normalization, tied input and output embeddings, and the 32K-token Llama tokenizer of TinyLlama (Zhang et al., 2024). A (Gated) DeltaNet layer uses nine heads with key dimension 128 and value dimension 256. Both Gated DeltaNet and DeltaNet use the output gate so that they have identical parameter counts and differ only in the learned decay. A dense OnlineTSSA operator uses nine dictionaries of 256 atoms each, implemented as nine recurrent heads of the same total width. A sparse OnlineTSSA operator uses 45 dictionaries, while each token activates only 11 dictionaries, each with 256 atoms; it matches the total number of parameters compared to the SwiGLU used in other models. In OnlineTSSA operators, each base dictionary $\mathbf{U}_{0,k}$ serves as the encoder and as the basis of the residual, and the decoder of the routed output is a separately learned matrix of the same shape.

Long-context tasks. *(i) Synthetic recall* tasks are of the "needle-in-a-haystack" nature: the goal is to retrieve an important piece of information from a long sequence of irrelevant information. *(ii) In retrieval-augmented generation (RAG)*, the model answers an open-domain question from retrieved Wikipedia passages, one of which contains the answer. *(iii) In many-shot in-context learning (ICL)*, the model classifies a text given labeled examples in the context, with the label names replaced by random integers so that the labels must be learned from the examples. *(iv) In long-document question answering (LongQA)*, the model answers a question from the paragraphs that contain the answer, placed among many other paragraphs of the same dataset. Synthetic recall is run at context lengths of 1K to 8K tokens, i.e., up to twice the training length, and the other three categories at 4K to 16K tokens, i.e., up to four times the training length. All tasks use greedy decoding. Since none of the models is instruction-tuned, we use the prompt format that each benchmark prescribes for base models, following Yang et al. (2025a).

Synthetic recall. We employ ten datasets from RULER (Hsieh et al., 2024): S-NIAH-2/3, MK-NIAH-1,2,3, MV-NIAH, MQ-NIAH, VT, CWE, FWE, with 500 samples per task and length. As discussed in recent work (Peng et al., 2026), the dataset S-NIAH-1 deviates significantly from natural language, hence we refrain from including it in Figure 2. That being said, Table 2 reports the results on all the above tasks including S-NIAH-1 for completeness.

RAG and many-shot ICL. Both categories follow HELMET (Yen et al., 2025), whose 8K and 16K settings we use as they are and whose 4K setting we obtain from the same data with proportionally fewer passages and demonstrations. In RAG, we use Natural Questions, TriviaQA, PopQA, and HotpotQA; each question comes with k passages of a Wikipedia dump ($k = 20$, 50, and 105 at 4K, 8K, and 16K): the gold passage and the top retrieved passages that do not contain the answer. We sample 100 questions per dataset and ask each with the gold passage at six positions in the context (for HotpotQA, its two gold passages in three random orders), which gives 600 samples per dataset and length (300 for HotpotQA). PopQA keeps the questions about entities with fewer than 10^3 monthly page views. The score is the substring exact match of HELMET, i.e., whether a

reference answer occurs in the generation after normalization. In ICL, we use TREC-coarse, TREC-fine, NLU, BANKING77, and CLINC150; each of 500 label-balanced test questions is preceded by label-balanced demonstrations from the training set (200 to 255 at 4K depending on the dataset, twice and four times as many at 8K and 16K), with the label names replaced by a random permutation of integers drawn per question. The score is the exact match of the predicted integer; chance is 16.7% for TREC-coarse, which has six labels, and below 2% for the other four datasets, which have 50 to 151.

LongQA. These are the two question-answering tasks of RULER, SQuAD and HotpotQA. A SQuAD or HotpotQA question is asked about its gold paragraphs, which are placed in random order among as many other paragraphs of the same dataset as fit the context length (20 at 4K, 40 to 50 at 8K, and 80 to 100 at 16K), with 500 samples per task and length. A sample is correct if one of its reference answers occurs in the generation, the rule of RULER; the implementation in the `lm-evaluation-harness` scores the fraction of the reference strings found instead, which we corrected.

Table 2: Score (%) on the synthetic recall tasks from RULER at context lengths of 1K to 8K tokens, 500 samples per task and length; all models are trained at length 4K. The score of a sample is the fraction of its reference strings found in the generation. Bold marks the best model for each task and length; higher is better.

	Transformer++				DeltaNet				Gated DeltaNet				FOLD (Ours)				FOLD-MLP (Ours)			
Task	**1K**	**2K**	**4K**	**8K**	**1K**	**2K**	**4K**	**8K**	**1K**	**2K**	**4K**	**8K**	**1K**	**2K**	**4K**	**8K**	**1K**	**2K**	**4K**	**8K**
S-NIAH-1	**100.0**	99.2	92.4	0.0	**100.0**	**100.0**	99.4	**100.0**	**100.0**	**100.0**	**99.8**	92.8	**100.0**	**100.0**	**99.8**	75.0	99.4	97.2	63.4	29.8
S-NIAH-2	**100.0**	**99.8**	**98.8**	0.0	**100.0**	84.8	25.8	11.6	**100.0**	**99.8**	58.4	25.4	**100.0**	62.0	76.2	16.6	99.2	99.0	83.6	**43.8**
S-NIAH-3	31.6	36.2	1.6	0.0	89.4	54.2	20.4	6.8	78.8	15.0	38.2	11.2	85.4	64.8	**69.2**	9.4	**89.6**	**72.4**	40.2	**16.4**
MK-NIAH-1	**65.6**	**75.2**	**65.0**	0.0	28.4	22.8	20.6	8.8	35.8	26.8	22.6	16.2	23.8	18.0	20.6	5.4	42.0	35.0	30.0	**22.2**
MK-NIAH-2	**15.6**	**7.2**	**3.2**	0.0	1.4	0.6	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.2	0.6	**0.2**
MK-NIAH-3	**24.8**	**7.8**	**5.8**	0.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
MV-NIAH	**43.1**	**42.2**	**31.6**	0.0	28.7	21.9	9.2	5.4	30.8	28.4	21.3	9.2	11.2	9.2	14.8	6.2	32.2	34.9	25.1	**20.4**
MQ-NIAH	**44.6**	**38.9**	**28.5**	0.0	23.3	23.3	10.5	4.5	26.9	26.1	20.8	7.1	10.8	15.6	18.0	8.4	23.6	21.4	20.5	**12.8**
VT	0.9	1.2	0.8	0.0	1.8	1.5	2.8	4.2	1.9	**3.0**	**3.1**	**4.5**	**2.7**	2.4	2.2	0.1	1.5	2.0	0.2	0.0
CWE	21.6	2.6	**55.8**	0.0	**58.1**	**25.3**	33.3	7.6	47.0	17.2	30.0	**11.2**	34.3	11.7	10.4	2.3	16.7	8.7	7.3	0.2
FWE	**37.8**	7.9	0.0	0.1	9.5	**8.5**	**9.1**	**6.1**	4.7	7.0	5.1	3.9	3.2	2.9	0.9	1.1	7.8	3.5	1.9	1.0
Mean w/o S-NIAH-1	**38.6**	**31.9**	**29.1**	0.0	34.1	24.3	13.2	5.5	32.6	22.3	19.9	8.9	27.2	18.7	21.2	5.0	31.7	27.7	20.9	**11.7**
Mean, all 11 tasks	**44.2**	**38.0**	**34.9**	0.0	40.1	31.2	21.0	14.1	38.7	29.4	27.2	**16.5**	33.8	26.1	28.4	11.3	37.8	34.0	24.8	13.3

Table 3: Score (%) on the RAG and many-shot ICL datasets of HELMET and on the two LongQA tasks of RULER at context lengths of 4K to 16K tokens; all models are trained at length 4K. RAG: substring exact match over 600 samples per dataset and length (300 for HotpotQA). ICL: accuracy over 500 samples; chance is 16.7 for TREC-coarse and below 2 for the others. LongQA: whether a reference answer occurs in the generation, 500 samples. Bold marks the best model for each row and length; higher is better.

	Transformer++			DeltaNet			Gated DeltaNet			FOLD (Ours)			FOLD-MLP (Ours)		
Dataset	**4K**	**8K**	**16K**	**4K**	**8K**	**16K**	**4K**	**8K**	**16K**	**4K**	**8K**	**16K**	**4K**	**8K**	**16K**
NQ	**14.3**	1.0	1.0	12.0	7.0	6.5	9.2	3.8	2.0	8.0	5.5	2.7	13.5	**11.0**	**7.0**
TriviaQA	**27.2**	1.2	1.7	21.7	14.2	14.3	18.5	15.2	11.0	19.0	11.2	13.3	23.3	**21.8**	**17.5**
PopQA	**20.5**	2.7	0.5	14.2	**16.0**	15.8	20.0	15.5	**17.5**	7.3	3.8	8.7	16.5	15.8	15.5
HotpotQA	15.0	1.0	0.0	**15.7**	**11.0**	**10.7**	10.7	7.0	3.7	9.3	7.0	5.0	9.3	9.3	7.7
RAG mean	**19.2**	1.5	0.8	15.9	12.0	11.8	14.6	10.4	8.5	10.9	6.9	7.4	15.7	**14.5**	**11.9**
TREC-coarse	**43.4**	18.6	17.8	19.6	16.8	18.2	20.0	20.8	22.0	24.0	22.0	22.0	24.6	**30.0**	**27.0**
TREC-fine	1.6	1.4	1.4	**2.0**	1.2	1.8	1.6	1.6	2.4	**2.0**	1.6	1.0	1.8	**2.4**	**2.8**
NLU	**5.0**	1.8	1.8	3.4	1.6	1.2	1.6	**2.6**	1.2	2.0	1.8	**2.2**	1.2	2.0	1.4
BANKING77	**2.8**	1.6	0.4	0.8	1.8	1.2	1.4	**2.2**	2.2	0.8	1.6	1.4	2.0	**2.2**	**2.4**
CLINC150	**1.4**	**1.0**	0.4	0.6	0.8	**1.6**	1.2	0.8	0.8	0.8	**1.0**	0.8	0.8	0.6	0.2
ICL mean	**10.8**	4.9	4.4	5.3	4.4	4.8	5.2	5.6	5.7	5.9	5.6	5.5	6.1	**7.4**	**6.8**
SQuAD	**16.6**	4.8	0.8	12.8	6.4	4.0	9.0	6.0	3.0	9.4	7.6	**4.8**	13.6	**7.8**	3.8
HotpotQA	**14.8**	2.6	0.2	10.2	11.0	8.0	10.0	8.0	7.4	13.8	**11.8**	**9.4**	11.4	9.8	8.0
LongQA mean	**15.7**	3.7	0.5	11.5	8.7	6.0	9.5	7.0	5.2	11.6	**9.7**	**7.1**	12.5	8.8	5.9

E.2 Ablation Studies

Routing. The router can be implemented as a learned linear softmax router or a dictionary-derived one as described in (19). We compare these two choices and also test sparse routing, in which only three of the nine dictionaries are active for each token. We train these variants while keeping the other settings identical to the FOLD-MLP tested in Section 4.1, and evaluate them on the SlimPajama validation set with a context length of 4096 ($1\times$) and 8192 ($2\times$).

Table 4: SlimPajama routing ablations. Bold values identify the better dense result at each evaluation length; lower is better.

Router	$1\times$	$2\times$
Learned, dense	**2.519**	2.483
Dictionary-derived, dense	2.520	**2.482**
Learned, 3 of 9 active	2.572	2.537

The two dense routers are effectively indistinguishable: their losses differ by only 0.001 at either length, and each is better at one evaluation length. This agreement supports the dictionary-derived router as a principled alternative to a separately learned router. Activating only three of nine dictionaries increases the loss by 0.053 at $1\times$ and 0.054 at $2\times$ relative to dense learned routing. Sparse routing therefore exposes a clear accuracy–computation trade-off, but the present results do not include throughput measurements and cannot quantify the corresponding efficiency gain.