FALCON: Fast-Weight Attention for Continual Learning

FWA Authors

March 9, 2026

Abstract

Recurrent fast-weight memories and selective state-space models compress a growing context into a bounded-size recurrent memory, so their update rule acts as an online continual-learning rule. We study this rule under a read-after-write autoregressive convention. Under this convention, the internal fast-memory objective that trains the state to support the next prediction is naturally defined on the prefix-aligned pair $(\mathbf{x}_t, \mathbf{y}_t) = (\phi(\mathbf{k}_{t-1}), \mathbf{v}_t)$. The same-step pairing $(\mathbf{k}_t, \mathbf{v}_t)$ is still causal, but it corresponds to a different local objective. We make this distinction explicit and derive normalized first-order updates for both regression and inner-product memories. The resulting family includes FALCON-2, an NLMS-stabilized delta rule with optional per-column gains; FALCON-3, a sliding-window mini-batch regression rule; and the inner-product objective variants FALCON-2A/FALCON-3A. We also show how these updates admit chunk-parallel unrollings compatible with SSD-style training. This view casts recurrent sequence models as fast continual learners with explicit controls for plasticity, forgetting, and bounded rehearsal.

Project Page: https://github.com/yifanzhang-pro/FALCON

1 Introduction

Transformers (Vaswani et al., 2017) stand as the ubiquitous backbone of modern Natural Language Processing, powering foundational Large Language Models (LLMs) such as GPT-4 (Achiam et al., 2023), Gemini (Team et al., 2023), and Llama (Touvron et al., 2023). The efficacy of the Transformer stems from the self-attention mechanism, which models global dependencies to generate rich, context-aware representations. However, this capability comes at a cost: standard attention scales quadratically $\mathcal{O}(N^2)$ with sequence length N. This complexity imposes severe bottlenecks for long-context processing, where the compute required for the attention matrix and the memory bandwidth for the Key-Value (KV) cache become prohibitive.

Beyond efficiency, long-context modeling is also a continual-learning problem: the model needs a *fast memory* that binds new evidence online while avoiding catastrophic interference from unbounded accumulation. Transformers externalize this fast memory as a growing key-value KV cache. SSMs and many fast-weight models, on the other hand, compress it into a bounded-size recurrent memory; in the strictly Markov case, this is a fixed-size state, while sliding-window variants additionally require a fixed-width tail. In this setting, the state-update rule functions as a local learning rule, and its temporal alignment determines whether the fast memory is trained on information that is

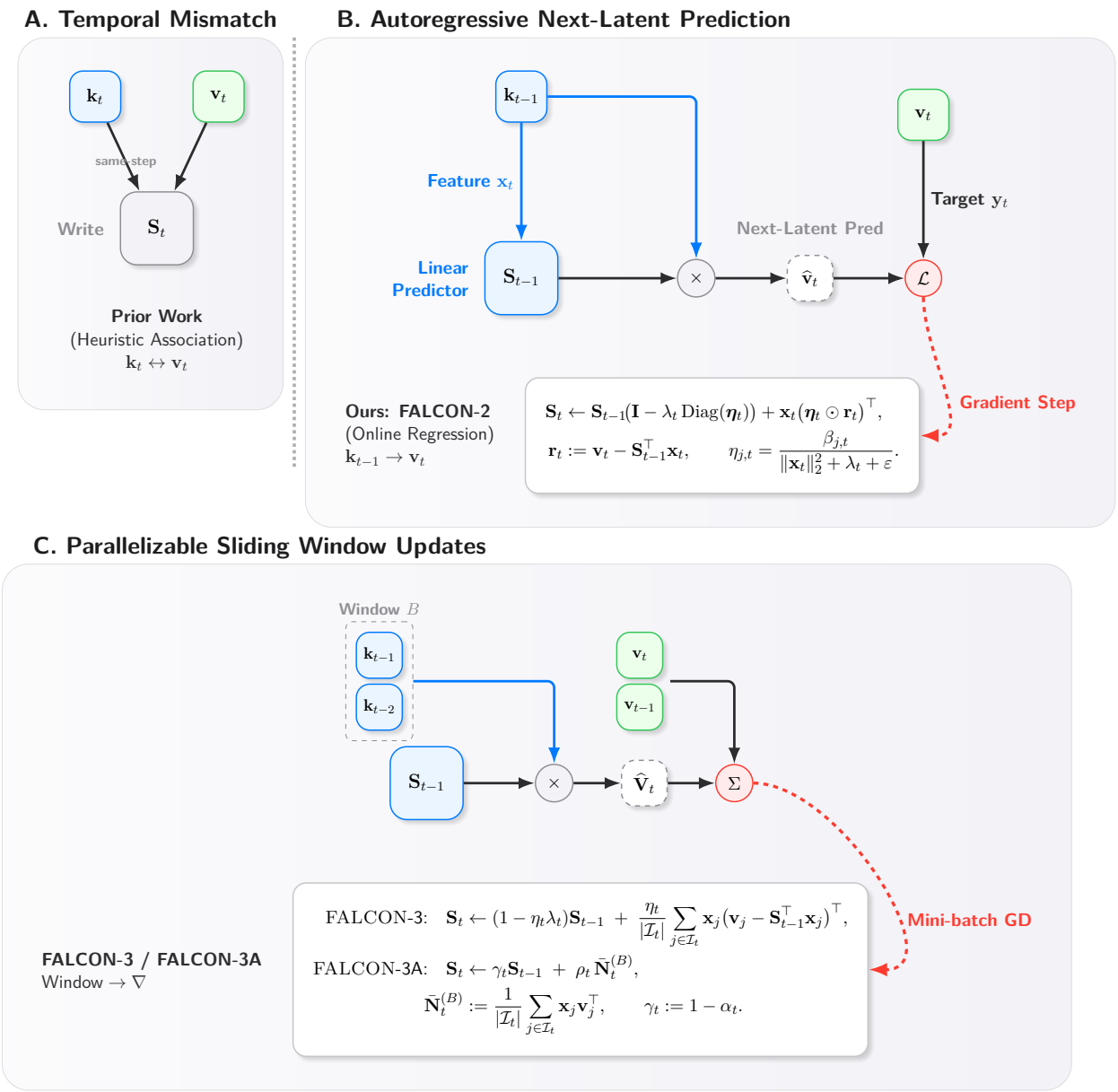


Figure 1 Conceptual Overview. (A) Many fast-weight rules bind $(\mathbf{k}_t, \mathbf{v}_t)$ (cache-style association). Under the *prefix-prediction* fast-memory objective studied here, the example revealed at step t pairs a prefix feature with the newly observed target, yielding $(\mathbf{k}_{t-1}, \mathbf{v}_t)$. The same-step association is still causal; it simply optimizes a different local predictor. (B) **FALCON-2 (Next-Latent Prediction):** the state $\mathbf{S}_{t-1}$ predicts $\mathbf{v}_t$ from the prefix feature $\mathbf{x}_t := \phi(\mathbf{k}_{t-1})$, then performs an online ridge-regression update. In full FALCON-2, the NLMS gains are *per column*: $\eta_{j,t} = \beta_{j,t}/(\|\mathbf{x}_t\|_2^2 + \lambda_t + \varepsilon)$, so both the update magnitude and the shrinkage factor $1 - \lambda_t\eta_{j,t}$ are column-wise (equivalently, per output feature / column of $\mathbf{S}_t$). This also keeps the notation consistent: $\beta_{j,t}$ is the gain, while $\eta_{j,t}$ is the actual normalized step size. (C) Sliding Regression (ATLAS (Behrouz et al., 2025a), **FALCON-3**) applies a mini-batch regression step over an active window $\mathcal{I}_t$ of nominal size B (realized size $B_t := |\mathcal{I}_t| \le B$) over causal pairs $(\mathbf{x}_j, \mathbf{v}_j)$ with $\mathbf{x}_j := \phi(\mathbf{k}_{j-1})$. Its inner-product counterpart **FALCON-3A** uses the window-average cross-covariance $\bar{\mathbf{N}}_t^{(B)}$.

actually available at prediction time (Sun et al., 2024; Liu et al., 2024a; Behrouz et al., 2024; Wang et al., 2025).

Architectures such as Linear Attention (Katharopoulos et al., 2020), Fast Weight Programmers (Schlag et al., 2021), RWKV (Peng et al., 2023), and Mamba (Gu and Dao, 2023) are competitive with Transformers on selected benchmarks at the small- to medium-scale regime while maintaining $\mathcal{O}(N)$ scaling in training and $\mathcal{O}(1)$ complexity per step during inference.

Despite these advances, the update rule governing the recurrent state, how the model writes to and forgets from memory, remains predominantly heuristic. Moreover, once one fixes the strict read-after-write semantics used in this paper, there is an easy convention mismatch: many recurrences are written under the standard Transformer convention and therefore bind $(\mathbf{k}_t, \mathbf{v}_t)$. Under the prefix-prediction local objective studied here, the example revealed at step t instead pairs the newly observed target $\mathbf{v}_t$ with a prefix feature computed at step $t-1$, i.e. $(\mathbf{k}_{t-1}, \mathbf{v}_t)$ or equivalently $(\mathbf{k}_i, \mathbf{v}_{i+1})$ under standard indexing. The same-step pairing is still causal; it simply corresponds to a different internal fast-memory objective. We make this convention dependence explicit throughout.

Titans and ATLAS view the recurrent state as a fast memory trained by optimizing an internal objective over the stream (Behrouz et al., 2024, 2025a). Our contribution is orthogonal: we make the *autoregressive* training pair explicit ($\mathbf{k}_{t-1} \to \mathbf{v}_t$ under read-after-write) and derive closed-form first-order updates that remain compatible with SSD-style chunk-parallel training (Dao and Gu, 2024). In particular, FALCON-3 uses a single mini-batch gradient step over a sliding window. Its regression form admits associative-scan / ParallelFlow unrollings, while its inner-product counterpart FALCON-3A admits a masked-attention unrolling. By contrast, many linear-attention/SSD formulations rely on purely additive inner-product writes.

In this work, we revisit the fundamentals of state-based modeling through the lens of *autoregressive next-latent prediction*. We formalize the state update as an explicit online optimization problem and make explicit the one-step shift required by our read-after-write next-latent convention. In our framework, the recurrent state matrix $\mathbf{S}_t$ serves as a linear predictor attempting to map the prefix write feature $\mathbf{x}_t := \phi(\mathbf{k}_{t-1})$ to $\mathbf{v}_t$. By analyzing SSMs and Linear Attention as online optimizers, we identify distinct objective functions, squared-error regression versus a negative inner-product objective (equivalently, inner-product maximization in the unregularized case), that drive their respective dynamics. Our contributions are as follows:

- We interpret the recurrent state as *fast weights* updated by an online learning rule. The gain β_t controls *plasticity* (Liu et al., 2024a), the ridge coefficient λ_t induces *forgetting*, and the sliding-window variants implement a bounded rehearsal signal that mitigates long-range interference.
- **FALCON-2:** We derive an improved Delta update rule from the Normalized Least Mean Squares (NLMS) algorithm. By introducing input-norm normalization together with per-column adaptive gains and decay, FALCON-2 explicitly accounts for varying input norms and heterogeneous output-feature dynamics while maintaining parallel efficiency.
- **FALCON-3:** We generalize the regression objective to a Sliding State mechanism via mini-batch SGD, yielding a finite-window optimization signal that mitigates long-range interference and noise accumulation with parallelizable algorithms for hardware-friendly training and inference. This is closely related to the Omega learning rule introduced in ATLAS (Behrouz et al., 2025a), which optimizes an internal memory objective over a sliding context window rather than only the most recent token.

- **FALCON-3A:** We introduce the inner-product counterpart to sliding states, combining chunk-parallel training with a sliding-window accumulation rule and a decoupled parameterization of plasticity and forgetting: a write coefficient ρ_t controls the injection magnitude, while a dimensionless decay fraction α_t controls forgetting.

2 Background

2.1 State Space Models

State Space Models (SSMs) (Gu and Dao, 2023; Dao and Gu, 2024) process a sequence of inputs $x(t) \in \mathbb{R}^{d_{\text{in}}}$ through a compressed latent state $\mathbf{h}(t) \in \mathbb{R}^n$, producing outputs $y(t) \in \mathbb{R}^{d_{\text{out}}}$. The continuous-time linear dynamics are governed by

$$\dot{\mathbf{h}}(t) = \mathbf{A}(t)\mathbf{h}(t) + \mathbf{B}(t)x(t), \qquad y(t) = \mathbf{C}(t)^\top \mathbf{h}(t),$$

where $\mathbf{B}(t) \in \mathbb{R}^{n \times d_{\text{in}}}$ and $\mathbf{C}(t) \in \mathbb{R}^{n \times d_{\text{out}}}$ (so $\mathbf{C}(t)^\top \mathbf{h}(t)$ is d_{out}-dimensional). To operate on discrete sequences, these dynamics are discretized over a step size Δ_t. Under the zero-order hold assumption, the resulting discrete-time system admits an exact closed-form solution.

To avoid conflating discretization schemes, we make the discrete-time mapping explicit. Over step size Δ_t, the standard form is

$$\mathbf{h}_t = \bar{\mathbf{A}}_t \mathbf{h}_{t-1} + \bar{\mathbf{B}}_t x_t, \qquad \bar{\mathbf{A}}_t = e^{\Delta_t \mathbf{A}_t}, \qquad \bar{\mathbf{B}}_t = \int_0^{\Delta_t} e^{(\Delta_t - s)\mathbf{A}_t} \mathbf{B}_t \, ds. \tag{2.1}$$

When $\mathbf{A}_t$ is diagonal, the input integral $\bar{\mathbf{B}}_t$ admits an elementwise closed form:

$$\bar{\mathbf{B}}_t = \left(\mathbf{A}_t^{-1}(e^{\Delta_t \mathbf{A}_t} - \mathbf{I})\right) \mathbf{B}_t,$$

interpreting $\mathbf{A}_t^{-1}(e^{\Delta_t \mathbf{A}_t} - \mathbf{I})$ elementwise and using the limit $(e^{\Delta a} - 1)/a \to \Delta$ as $a \to 0$. A first-order approximation is $\bar{\mathbf{B}}_t \approx \Delta_t \mathbf{B}_t$ (valid when $\|\Delta_t \mathbf{A}_t\|$ is small), yielding

$$\mathbf{h}_t \approx e^{\Delta_t \mathbf{A}_t} \mathbf{h}_{t-1} + \Delta_t \mathbf{B}_t x_t.$$

Modern selective SSMs, such as Mamba (Gu and Dao, 2023), parametrize the system matrices $(\mathbf{B}, \mathbf{C}, \Delta)$ as functions of the current input x_t, allowing for content-aware filtering. Mamba-2 (Dao and Gu, 2024) further simplifies the transition matrix $\mathbf{A}$ to a scalar or diagonal structure, establishing a theoretical bridge known as Structured State Space Duality (SSD). This duality demonstrates that the recurrent SSM is equivalent to a specific form of causal linear attention, enabling efficient training via chunk-parallel matrix multiplications while maintaining constant-state inference.

2.2 Linear Attention

Linear Attention (Katharopoulos et al., 2020) circumvents the $\mathcal{O}(N^2)$ complexity of standard attention by replacing the softmax with a kernel feature map $\phi(\cdot) : \mathbb{R}^d \to \mathbb{R}^m$ such that $\kappa(\mathbf{q}_t, \mathbf{k}_j) = \phi(\mathbf{q}_t)^\top \phi(\mathbf{k}_j)$. Exploiting the associativity of matrix multiplication, the output $\mathbf{y}_t \in \mathbb{R}^{d_v}$ for the t-th token is:

$$\mathbf{y}_t = \frac{\sum_{j=1}^t \phi(\mathbf{q}_t)^\top \phi(\mathbf{k}_j) \mathbf{v}_j}{\sum_{j=1}^t \phi(\mathbf{q}_t)^\top \phi(\mathbf{k}_j)} = \frac{\left(\sum_{j=1}^t \phi(\mathbf{k}_j) \mathbf{v}_j^\top\right)^\top \phi(\mathbf{q}_t)}{\phi(\mathbf{q}_t)^\top \sum_{j=1}^t \phi(\mathbf{k}_j)}, \tag{2.2}$$

This formulation allows the context to be compressed into a recurrent matrix state $\mathbf{S}_t \in \mathbb{R}^{m \times d_v}$ and a normalizer $\mathbf{z}_t \in \mathbb{R}^m$:

$$\mathbf{y}_t = \frac{\mathbf{S}_t^\top \phi(\mathbf{q}_t)}{\mathbf{z}_t^\top \phi(\mathbf{q}_t) + \varepsilon_{\text{attn}}}, \qquad \mathbf{S}_t = \mathbf{S}_{t-1} + \phi(\mathbf{k}_t)\mathbf{v}_t^\top, \quad \mathbf{z}_t = \mathbf{z}_{t-1} + \phi(\mathbf{k}_t). \tag{2.3}$$

Here $\varepsilon_{\text{attn}} \geq 0$ is a small stabilizer. The normalized form is intended only when the read normalizer is nonnegative, e.g. for positive feature maps with $\phi(\cdot) \geq 0$ elementwise, so that $\mathbf{z}_t^\top \phi(\mathbf{q}_t) \geq 0$. In signed-feature settings (including the identity map without an additional positivity constraint), $\varepsilon_{\text{attn}}$ only prevents division by zero and does not make the denominator attention-like; in that case, the denominator-free form introduced below is the mathematically relevant default.

Causality and indexing. Eq. (2.3) follows the standard Transformer convention: at position t we read from the updated state $(\mathbf{S}_t, \mathbf{z}_t)$, and the resulting representation is used to predict token $t{+}1$. Our *next-latent* alignment shifts the write stream by one: after observing $\mathbf{v}_t$ we write it under the previous key $\mathbf{k}_{t-1}$, or equivalently, $(\mathbf{k}_i, \mathbf{v}_{i+1})$ under standard indexing with $i = t - 1$. This yields the read-after-write recurrence

$$\mathbf{y}_t = \frac{\mathbf{S}_t^\top \phi(\mathbf{q}_t)}{\mathbf{z}_t^\top \phi(\mathbf{q}_t) + \varepsilon_{\text{attn}}}, \qquad \mathbf{S}_t = \mathbf{S}_{t-1} + \phi(\mathbf{k}_{t-1})\mathbf{v}_t^\top, \quad \mathbf{z}_t = \mathbf{z}_{t-1} + \phi(\mathbf{k}_{t-1}), \tag{2.4}$$

where $\varepsilon_{\text{attn}} \geq 0$ is a small stabilizer. Defining the shifted key stream $\widetilde{\mathbf{k}}_t := \mathbf{k}_{t-1}$, Eq. (2.4) matches Eq. (2.3) after replacing $\mathbf{k}_t$ with $\widetilde{\mathbf{k}}_t$; the substantive change is the *key-value alignment* $(\widetilde{\mathbf{k}}_t, \mathbf{v}_t)$.

Normalized vs. unnormalized linear attention. The denominator in Eq. (2.2) uses the normalizer state $\mathbf{z}_t$ to rescale the readout. Many SSM/SSD-style architectures instead drop the denominator (and $\mathbf{z}_t$) and use an *unnormalized* inner-product read:

$$\mathbf{y}_t = \mathbf{S}_t^\top \phi(\mathbf{q}_t), \qquad \mathbf{S}_t = (1 - \eta_t \lambda_t)\mathbf{S}_{t-1} + \eta_t \, \phi(\mathbf{k}_{t-1})\mathbf{v}_t^\top, \tag{2.5}$$

with the unshifted convention recovered by replacing $\mathbf{k}_{t-1}$ with $\mathbf{k}_t$. In this denominator-free form, bounded state magnitude and a controllable memory timescale are enforced by explicit decay (e.g., $\lambda_t > 0$) and/or gain control. Section 4.3 shows that the additive update is exactly gradient descent on an inner-product objective.

This recurrence is the gradient-descent update for the inner-product objective in Section 4.3. When $\lambda_t = 0$, the write is *purely additive* rank-one Hebbian learning; when $\lambda_t > 0$, it is additive plus scalar shrinkage. As noted in Section 2.1, Mamba-2 proves that the no-normalizer recurrence is functionally equivalent to a specific class of SSMs under the SSD framework.

2.3 Delta Networks

Fast Weight Programmers and Delta Networks (Schlag et al., 2021) formulate sequence modeling as the online learning of a value-retrieval function. Let $\mathbf{S}_{t-1} \in \mathbb{R}^{d \times d_v}$ denote the fast-weight state matrix. Instead of purely additive accumulation, standard Delta Networks employ an error-driven update derived from the gradient of the instantaneous squared error between the state's reconstruction of the *current* key and value:

$$\ell_t(\mathbf{S}) := \frac{1}{2} \left\| \mathbf{S}^\top \mathbf{k}_t - \mathbf{v}_t \right\|_2^2. \tag{2.6}$$

Here, $\mathbf{S}^\top \mathbf{k}_t$ represents the model's prediction of value $\mathbf{v}_t$ given key $\mathbf{k}_t$. The gradient with respect to the state is $\nabla_{\mathbf{S}} \ell_t(\mathbf{S}) = \mathbf{k}_t(\mathbf{S}^\top \mathbf{k}_t - \mathbf{v}_t)^\top$.

Step-size convention. We denote the gradient step size by η_t (prior DeltaNet literature often writes this quantity as β_t). This keeps notation consistent with Section 3, where β_t denotes an NLMS *gain* and η_t the resulting step size. A single step of online Gradient Descent with step size η_t yields the Delta rule:

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \eta_t \, \mathbf{k}_t(\mathbf{S}_{t-1}^\top \mathbf{k}_t - \mathbf{v}_t)^\top = (\mathbf{I} - \eta_t \mathbf{k}_t \mathbf{k}_t^\top)\mathbf{S}_{t-1} + \eta_t \mathbf{k}_t \mathbf{v}_t^\top. \tag{2.7}$$

The rank-one factor $(\mathbf{I} - \eta_t \mathbf{k}_t \mathbf{k}_t^\top)$ performs a targeted shrinkage/edit along the current key direction. It becomes an orthogonal projection onto the subspace orthogonal to $\mathbf{k}_t$ only in the special case $\eta_t = 1/\|\mathbf{k}_t\|_2^2$. This contrasts with the purely additive updates of Linear Attention.

Gated Delta Networks. Recent work on Gated Delta Networks (Yang et al., 2024a) observes that the implicit forgetting of the delta rule depends entirely on the alignment between $\mathbf{k}_t$ and the state, which may be insufficient for rapid context switching. They unify Mamba-style gating with the delta rule by introducing an explicit decay gate $\alpha_t \in [0, 1]$:

$$\mathbf{S}_t = g_t(\mathbf{I} - \eta_t \mathbf{k}_t \mathbf{k}_t^\top)\mathbf{S}_{t-1} + \eta_t \mathbf{k}_t \mathbf{v}_t^\top, \tag{2.8}$$

where we write $g_t \in [0, 1]$ (the gate denoted α_t in Yang et al. (2024a)) to avoid overloading notation. This gated Delta rule provides two complementary mechanisms: g_t allows for global state decay (similar to SSMs), while the rank-one term $(\mathbf{I} - \eta_t \mathbf{k}_t \mathbf{k}_t^\top)$ performs targeted edits to key-value associations.

3 Autoregressive Next-Latent Prediction

In this section, we formalize the recurrent state update not as a heuristic accumulation of history, but as an explicit optimization problem. Under the read-after-write convention adopted here, the online objective must be temporally aligned with next-token prediction. Standard Delta Networks minimize $\|\mathbf{S}^\top \mathbf{k}_t - \mathbf{v}_t\|^2$, i.e., they train on the pair $(\mathbf{k}_t, \mathbf{v}_t)$. Under our convention, the causal training example revealed at step t pairs the newly observed target $\mathbf{v}_t$ with a prefix feature that existed when $\mathbf{v}_t$ was predicted (at step $t-1$). This yields the next-latent pairing $\mathbf{k}_{t-1} \to \mathbf{v}_t$ (equivalently, $\mathbf{k}_i \to \mathbf{v}_{i+1}$ under standard Transformer indexing).

We posit that the hidden state $\mathbf{S}$ serves as a linear predictor trained online to map $\mathbf{k}_{t-1} \to \mathbf{v}_t$. Consider a sequence of keys $\{\mathbf{k}_t\}_{t=1}^T$ and values $\{\mathbf{v}_t\}_{t=1}^T$. We formulate the state update as an online ridge regression problem (Behrouz et al., 2024; Wang et al., 2025; Behrouz et al., 2025a).

At step t, the write feature must be computable from the prefix up to $t-1$, while the write target becomes available only after step t is revealed. Our *next-latent* alignment therefore pairs the previous key with the current value, $\mathbf{x}_t \triangleq \phi(\mathbf{k}_{t-1})$ and $\mathbf{y}_t \triangleq \mathbf{v}_t$ (with ϕ the identity in the non-kernelized case), and updates $\mathbf{S}$ by an online ridge regression step (Wang et al., 2025; Behrouz et al., 2024, 2025a). The instantaneous regularized least-squares objective at step t is:

$$\ell_t(\mathbf{S}) \triangleq \frac{1}{2} \left\| \mathbf{S}^\top \mathbf{x}_t - \mathbf{y}_t \right\|_2^2 + \frac{\lambda_t}{2} \|\mathbf{S}\|_F^2, \tag{3.1}$$

where $\lambda_t \geq 0$ is a regularization coefficient. While full-batch minimization of the cumulative loss corresponds to the offline solution found in methods like MesaNet (von Oswald et al., 2025), efficient

autoregressive modeling requires an online approximation. We therefore employ Online Gradient Descent (OGD).

Notation. In kernelized linear attention, the key that writes to memory is typically a feature vector $\phi(\mathbf{k}) \in \mathbb{R}^m$ rather than the raw key $\mathbf{k} \in \mathbb{R}^d$. All derivations in this section are interpreted

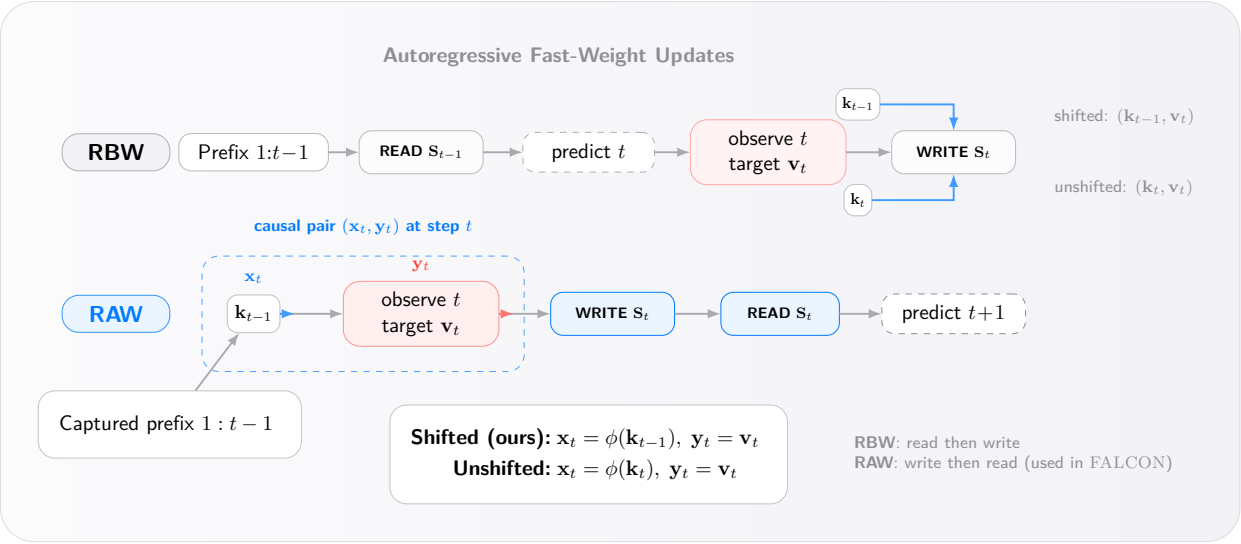
$$\mathbf{x}_t \equiv \phi(\mathbf{k}_{t-1}) \in \mathbb{R}^m, \qquad \mathbf{S}_t \in \mathbb{R}^{m \times d_v},$$

so we treat $\mathbf{x}_t$ as the generic *write feature* (the vector that appears in the update and in the normalizer) and keep notation uncluttered. Queries used for retrieval, e.g., $\phi(\mathbf{q}_t)$ in linear attention, live in the same feature space but need not equal $\mathbf{x}_t$.

Implicit fast-memory objective. Eq. (3.1) is the instantaneous objective whose gradient step *defines* the fast-memory write rule; it is not an additional supervised loss beyond the outer autoregressive likelihood. During training, we differentiate through the update so that the slow weights (which produce $(\mathbf{q}, \mathbf{k}, \mathbf{v})$ as well as β_t, λ_t) learn representations and gates that make these local updates useful.

Indexing and causality conventions. We index time by the current context position: at position t, the model has consumed tokens $1{:}t$ and predicts token $t{+}1$. We use a read-after-write order: after observing $\mathbf{y}_t$, we update the state to $\mathbf{S}_t$, and then read from $\mathbf{S}_t$ to form the representation at position t. Under next-latent alignment, the update at time t uses the causal pair $(\mathbf{k}_{t-1}, \mathbf{v}_t)$ (equivalently, $(\mathbf{k}_i, \mathbf{v}_{i+1})$ under standard Transformer indexing with $i = t-1$). The four cells in Fig. 2 correspond to different local-objective conventions. Under the RAW convention adopted here, the shifted pairing $(\mathbf{k}_{t-1}, \mathbf{v}_t)$ is the one aligned with the prefix feature that was available when $\mathbf{v}_t$ was predicted. The unshifted RAW pairing $(\mathbf{k}_t, \mathbf{v}_t)$ is still causal for next-token prediction, but it optimizes a different local objective based on same-step features rather than the prefix feature. Likewise, the shifted RBW pairing is not itself inconsistent, but it trains the memory after the read that produced the prediction at position t, so it does not match the internal predictor analyzed in this paper. We initialize $\mathbf{S}_0 = \mathbf{0}$ and adopt the paper-wide boundary convention $\mathbf{x}_1 := \mathbf{0}$ (equivalently, $\phi(\mathbf{k}_0) = \mathbf{0}$ for a zero BOS write feature). Under this default zero-state prefill, the $t = 1$ write is a no-op. If a nonzero state is carried across segments and one wants the boundary step to be a strict no-op even when $\lambda_1 > 0$, set $\eta_1 := 0$ as well. For simplicity, we write recursions for $t \geq 2$. With this convention, the next-latent prediction at step t is $\widehat{\mathbf{y}}_t := \mathbf{S}_{t-1}^\top \mathbf{x}_t$ and the update uses the residual $\mathbf{r}_t = \mathbf{y}_t - \widehat{\mathbf{y}}_t$. This $\widehat{\mathbf{y}}_t$ is the *internal fast-memory prediction* used by the local write loss; it should not be conflated with the model readout at position t, which under read-after-write is formed from the updated state (e.g. $\mathbf{o}_t = \mathbf{S}_t^\top \mathbf{q}_t$ in the denominator-free case). Equivalently, $\widehat{\mathbf{y}}_t$ is the prediction available at the end of step $t{-}1$ from the then-current state $\mathbf{S}_{t-1}$ and prefix feature $\mathbf{x}_t = \phi(\mathbf{k}_{t-1})$; after $\mathbf{y}_t$ is revealed we update to $\mathbf{S}_t$, and the readout at position t uses $\mathbf{S}_t$ to predict token $t{+}1$. In implementations, we optionally apply a short causal convolution before forming $(\mathbf{q}, \mathbf{k}, \mathbf{v})$, which improves local associative recall in line with prior findings (Fu et al., 2022; Liu et al., 2024b).

Fast memory as continual learning. The state $\mathbf{S}$ is a *fast* memory (fast weights) updated online within the forward pass, while the slow weights learn representations and gates that make these local updates useful. From a continual-learning lens, each token provides a new training example $(\mathbf{x}_t, \mathbf{y}_t)$: the write gain controls *plasticity*, the shrinkage/decay path controls *forgetting*, and the sliding-window variants in Section 4 implement bounded rehearsal to reduce long-range interference. For regression-style variants, this forgetting is naturally expressed via a ridge coefficient λ_t; for



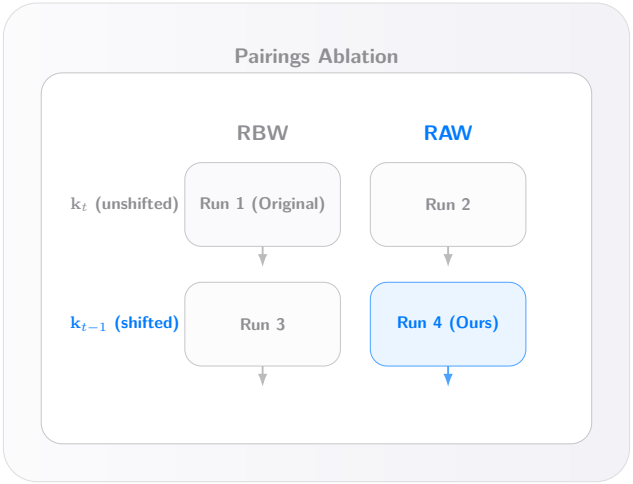


Figure 2 Formal timing conventions. Above: We explicitly distinguish *read-before-write* (RBW) and *read-after-write* (RAW) autoregressive fast-weight timing semantics. Under the RAW convention used in FALCON, the causally aligned training pair at step t is $(\mathbf{k}_{t-1}, \mathbf{v}_t)$, i.e., a prefix feature paired with the newly revealed target. Below: A 2×2 ablation over timing convention $\{\text{RBW}, \text{RAW}\}$ and write pairing $\{\mathbf{k}_t, \mathbf{k}_{t-1}\}$ examines whether gains arise from causal timing alignment rather than indexing alone.

the inner-product sliding-state variant FALCON-3A we instead parameterize a dimensionless decay fraction directly (Section 4.5), which is the quantity that actually governs the recurrence. Normalization in Eqs. (3.4), (4.8), and (4.18) is the key stabilizer, making the updates scale-robust across heterogeneous feature norms and preventing catastrophic drift over long sequences.

Identity matrices. We write $\mathbf{I}_{d_x}$ for the $d_x \times d_x$ identity in the feature space, where $d_x \triangleq \dim(\mathbf{x}_t)$ (so $d_x = d$ for raw keys and $d_x = m$ for kernelized write-features), and $\mathbf{I}_{d_v}$ for the $d_v \times d_v$ identity in value space. When the dimension is unambiguous, we write $\mathbf{I}$.

3.1 Online Gradient Descent Update

The gradient of the instantaneous loss in Eq. (3.1) with respect to the state $\mathbf{S}$ is:

$$\nabla_{\mathbf{S}} \ell_t(\mathbf{S}) = \mathbf{x}_t(\mathbf{S}^\top \mathbf{x}_t - \mathbf{y}_t)^\top + \lambda_t \mathbf{S}. \tag{3.2}$$

Applying a single gradient descent step with learning rate η_t yields the update rule:

$$\begin{aligned} \mathbf{S}_t &\leftarrow \mathbf{S}_{t-1} - \eta_t \nabla_{\mathbf{S}} \ell_t(\mathbf{S}_{t-1}) \\ &= \mathbf{S}_{t-1} - \eta_t \left[\mathbf{x}_t(\mathbf{S}_{t-1}^\top \mathbf{x}_t - \mathbf{y}_t)^\top + \lambda_t \mathbf{S}_{t-1} \right] \\ &= (1 - \eta_t \lambda_t)\mathbf{S}_{t-1} + \eta_t \mathbf{x}_t \mathbf{r}_t^\top, \end{aligned} \tag{3.3}$$

where $\mathbf{r}_t \triangleq \mathbf{y}_t - \mathbf{S}_{t-1}^\top \mathbf{x}_t$ is the residual (prediction error). Note that, unlike standard Delta Networks, $\mathbf{r}_t$ measures the discrepancy between the prediction from the previous key $\mathbf{k}_{t-1}$ and the current value $\mathbf{v}_t$.

This is gradient descent on the instantaneous ridge objective. With the normalized step size below, it becomes a normalized update. In the special case $\lambda_t = 0$ and $\varepsilon = 0$, it reduces exactly to the classical NLMS recursion, for $\varepsilon > 0$ it is the usual stabilized NLMS variant. Specifically, the loss ℓ_t is L_t-smooth with respect to the Frobenius norm, with smoothness constant $L_t = \|\mathbf{x}_t\|_2^2 + \lambda_t$. To ensure numerical stability and scale robustness, we adopt the normalized step size:

$$\eta_t = \frac{\beta_t}{\|\mathbf{x}_t\|_2^2 + \lambda_t + \varepsilon}, \qquad \beta_t \in (0, 2),\ \varepsilon \geq 0. \tag{3.4}$$

We adopt the convention $\eta_t := 0$ when $\|\mathbf{x}_t\|_2^2 + \lambda_t + \varepsilon = 0$. In analysis, we may take $\varepsilon = 0$ and assume $L_t > 0$ (equivalently, $\|\mathbf{x}_t\|_2 > 0$ or $\lambda_t > 0$). In implementations, we take $\varepsilon > 0$ for numerical robustness; any $\varepsilon > 0$ only decreases η_t and therefore preserves the descent guarantees below.

Gain vs. step size. We use $\beta_t \in (0, 2)$ to denote the dimensionless NLMS gain and η_t for the resulting (normalized) step size. Some prior DeltaNet works use β_t to denote the raw step size, our notation separates these two quantities. More precisely, in the unclamped $\varepsilon = 0$ analysis, $\beta_t > 1$ induces a negative eigenvalue along the current write-feature direction (Section 3), this regime can improve state-tracking behavior (Grazzi et al., 2024).

Interpreting ridge as decay. For $\lambda_t > 0$, the ridge term induces the scalar factor $\gamma_t := 1 - \eta_t \lambda_t$ in Eq. (3.3). However, the full linear transition is $\mathbf{A}_t = \gamma_t \mathbf{I}_{d_x} - \eta_t \mathbf{x}_t \mathbf{x}_t^\top$: it has eigenvalue γ_t on the subspace orthogonal to $\mathbf{x}_t$ and eigenvalue $\gamma_t - \eta_t \|\mathbf{x}_t\|_2^2 = 1 - \eta_t(\|\mathbf{x}_t\|_2^2 + \lambda_t)$ along $\mathbf{x}_t$. With $\varepsilon = 0$ and the NLMS step size Eq. (3.4), this directional eigenvalue equals $1 - \beta_t$, so $\beta_t > 1$ yields a (stable)

negative eigenvalue even if $\gamma_t > 0$. When log-space unrolling requires $\gamma_t > 0$, we implement the recurrence with

$$\alpha_t := \min(\eta_t \lambda_t,\, 1 - \varepsilon_\gamma), \qquad \gamma_t := 1 - \alpha_t,$$

and compute $\log \gamma_t = \mathrm{log1p}(-\alpha_t)$ in fp32. All exact renormalization identities below are exact for this implemented recurrence; when the clamp is inactive ($\eta_t \lambda_t < 1 - \varepsilon_\gamma$), it coincides with the original $\gamma_t = 1 - \eta_t \lambda_t$. The OGD/descent statements in this section refer to the *unclamped* recurrence. If the clamp activates, the implemented update is no longer exactly gradient descent on Eq. (3.1) with ridge coefficient λ_t; instead, it should be interpreted as using an *effective* decay coefficient $\widetilde{\lambda}_t := \alpha_t / \eta_t \le \lambda_t$ in the shrinkage path (with the convention $\widetilde{\lambda}_t := 0$ when $\eta_t = 0$). In the default RMS-normalized regime, the clamp is intended to remain inactive and serves only as a numerical safeguard.

3.2 Analysis

We show that the normalized step size yields per-step descent in the instantaneous *regularized objective* ℓ_t, a basic local stability property. This statement is pointwise in t: it does not imply monotone decrease of the cumulative online loss $\sum_s \ell_s$ or of the outer autoregressive training objective.

Lemma 3.1 (Per-step Descent for Smooth Losses)**.** Let $f : \mathbb{R}^{d_x \times d_v} \to \mathbb{R}$ be L-smooth with respect to the Frobenius norm. For any step size $\eta \in (0, 2/L)$, the gradient step $\mathbf{S}^+ = \mathbf{S} - \eta \nabla f(\mathbf{S})$ satisfies

$$f(\mathbf{S}^+) \le f(\mathbf{S}) - \frac{\eta(2 - \eta L)}{2} \|\nabla f(\mathbf{S})\|_F^2. \tag{3.5}$$

Proof. By L-smoothness, for any $\mathbf{S}$ and $\mathbf{S}'$,

$$f(\mathbf{S}') \le f(\mathbf{S}) + \langle \nabla f(\mathbf{S}), \mathbf{S}' - \mathbf{S} \rangle + \frac{L}{2} \|\mathbf{S}' - \mathbf{S}\|_F^2.$$

Set $\mathbf{S}' = \mathbf{S}^+ = \mathbf{S} - \eta \nabla f(\mathbf{S})$ and simplify. □

Step-size parametrization. Choosing $\eta = \beta / L$ with $\beta \in (0, 2)$ (hence requiring $L > 0$) yields a decrease coefficient $\beta(2 - \beta)/(2L)$. For $L > 0$, the stabilized choice $\eta = \beta/(L + \varepsilon)$ with $\varepsilon \ge 0$ also lies in $(0, 2/L)$. When $L = 0$, this interval is undefined; in the degenerate cases arising here ($\mathbf{x}_t = \mathbf{0}$ and $\lambda_t = 0$, or the corresponding windowed analogue), the gradient is zero, so we define the update to be a no-op by setting $\eta := 0$.

3.3 Delta Networks as Regression

In this section, we interpret many previous *fast weight* models under this optimization framework. We observe that Delta Networks and Linear Attention are not merely architectural heuristics, but solutions to specific online objective functions.

By substituting the regression assignments $\mathbf{x}_t \leftarrow \phi(\mathbf{k}_{t-1})$ and $\mathbf{y}_t \leftarrow \mathbf{v}_t$, Eq. (3.3) recovers the functional form of the Delta Network update rule (Schlag et al., 2021), but with the critical index shift:

$$\mathbf{S}_t = \underbrace{\left((1 - \eta_t \lambda_t) \mathbf{I}_{d_x} - \eta_t \mathbf{x}_t \mathbf{x}_t^\top\right)}_{\text{Decay \& Targeted Forget}} \mathbf{S}_{t-1} + \underbrace{\eta_t \mathbf{x}_t \mathbf{y}_t^\top}_{\text{Write}}, \qquad \mathbf{x}_t = \phi(\mathbf{k}_{t-1}),\ \mathbf{y}_t = \mathbf{v}_t. \tag{3.6}$$

Algorithm 1 Online ridge via SGD with batch size 1

Require: Keys $\{\mathbf{k}_t\}_{t=0}^{T}$ with zero-BOS write feature $\phi(\mathbf{k}_0) = \mathbf{0}$, feature map ϕ (default identity), values $\{\mathbf{v}_t\}_{t=1}^{T}$, ridge coefficients $\{\lambda_t\}_{t=1}^{T}$ with $\lambda_t \geq 0$, gain $\beta_t \in (0, 2)$, stabilizer $\varepsilon \geq 0$ (set $\eta_t{:=}0$ if the denominator is 0), init $\mathbf{S}_0 = \mathbf{0}$.

for $t = 1, \ldots, T$ **do**

  $\mathbf{x}_t \leftarrow \phi(\mathbf{k}_{t-1})$ ▷ Write feature (prefix, previous key)

  $\mathbf{y}_t \leftarrow \mathbf{v}_t$ ▷ Target (Current Value)

  $\widehat{\mathbf{y}}_t \leftarrow \mathbf{S}_{t-1}^{\top}\mathbf{x}_t$ ▷ Fast-memory prediction

  $\mathbf{r}_t \leftarrow \mathbf{y}_t - \widehat{\mathbf{y}}_t$ ▷ Residual (prediction error)

  $\eta_t \leftarrow 0$ if $t = 1$ or $\|\mathbf{x}_t\|_2^2 + \lambda_t + \varepsilon = 0$, else $\beta_t/(\|\mathbf{x}_t\|_2^2 + \lambda_t + \varepsilon)$ ▷ Normalized step size; this enforces the paper-wide boundary no-op

  $\mathbf{S}_t \leftarrow (1 - \eta_t\lambda_t)\mathbf{S}_{t-1} + \eta_t\, \mathbf{x}_t\mathbf{r}_t^{\top}$ ▷ Update State

end for

Here, the rank-one term $\mathbf{x}_t\mathbf{x}_t^{\top}\mathbf{S}_{t-1}$ is the left Hessian action of the squared-error loss along the current write-feature direction. Intuitively, it reduces the component of the current predictor that acts on $\mathbf{x}_t = \phi(\mathbf{k}_{t-1})$ before adding the new target $\mathbf{y}_t = \mathbf{v}_t$.

In contrast, if we replace the regression (MSE) loss with an inner-product objective (Section 4.3), the residual term disappears, and the update becomes purely additive. With the *standard* (unshifted) assignment $(\mathbf{x}_t, \mathbf{y}_t) = (\phi(\mathbf{k}_t), \mathbf{v}_t)$ (or $(\mathbf{k}_t, \mathbf{v}_t)$ in the unkernelized case) this reduces to the familiar Linear Attention / Mamba-2 accumulation; with our *next-latent* assignment $(\mathbf{x}_t, \mathbf{y}_t) = (\phi(\mathbf{k}_{t-1}), \mathbf{v}_t)$ it yields the one-step-shifted variant used by our methods (e.g., FALCON-3A in Section 4.5).

This regression perspective motivates a key algorithmic improvement that we analyze in Section 4: with NLMS-style normalization, the effective step size scales as $\eta_t \propto (\|\mathbf{x}_t\|_2^2 + \lambda_t)^{-1}$ (reducing to $\|\mathbf{x}_t\|_2^{-2}$ when $\lambda_t = 0$). This suggests that fixed learning rates are scale-mismatched for regression-style fast-weight updates, for inner-product writes, the same normalization is best viewed as a magnitude stabilizer rather than a curvature requirement.

4 FALCON: Fast Weight Attention

Fast-weight (matrix-valued) memories and linear attention transformers are widely viewed as online models that update a recurrent memory in the forward pass, tracing back to classic fast-weight memory mechanisms and their modern instantiations in linear Transformers and Delta-style rules in Hinton and Plaut (1987); Schmidhuber (1992); Ba et al. (2016); Schlag et al. (2021). In this section, we analyze State Space Models and Linear Attention through the lens of online optimization. We distinguish between two primary objective functions: regression (squared error) and a negative inner-product objective (equivalently, inner-product maximization in the unregularized case). By deriving their respective online update rules under our shifted $\mathbf{k}_{t-1} \to \mathbf{v}_t$ framework, we motivate specific improvements, including adaptive learning rates and sliding-window formulations.

4.1 Scaled Linear Attention and Scaled DeltaNet

Unlike softmax attention, which uses the scaled dot product $\langle \mathbf{q}, \mathbf{k} \rangle / \sqrt{d}$, fast-weight recurrences are directly sensitive to the norms of queries and keys: (i) dot-product reads grow with $\|\mathbf{q}_t\|_2\|\mathbf{k}\|_2$, and

(ii) additive (inner-product) writes grow with the write-feature norm. To stabilize both the *read* and the *write* streams, especially under long decoding horizons and mixed precision, we use explicit feature scaling/normalization.

Scaled features. We define a generic RMS normalization operator for a vector $\mathbf{u} \in \mathbb{R}^{d_u}$:

$$\mathrm{RMSNorm}(\mathbf{u}) := \frac{\mathbf{u}}{\mathrm{RMS}(\mathbf{u}) + \varepsilon_{\mathrm{rms}}}, \qquad \mathrm{RMS}(\mathbf{u}) := \frac{1}{\sqrt{d_u}}\|\mathbf{u}\|_2,$$

where $\varepsilon_{\mathrm{rms}} > 0$ is a small stabilizer. Unless stated otherwise, we apply RMSNorm to the $(\mathbf{q}_t, \mathbf{k}_t)$ projections used by fast-weight reads/writes, before forming dot products or outer products. This default differs from common ℓ_2-normalized DeltaNet variants (e.g., in Gated DeltaNet implementations) and is substantially more stable in mixed precision because RMSNorm keeps coordinate magnitudes $\Theta(1)$ while preserving the NLMS interpretation, since RMSNorm implies $\|\mathbf{u}\|_2^2 \approx d_u$. By default, we do *not* normalize values $\mathbf{v}_t$; value normalization (VNorm) is optional and disabled unless explicitly enabled.

Scaled Linear Attention. Under next-latent alignment, we use the RMS-normalized projections in the (optionally kernelized) feature space. The unnormalized readout is

$$\mathbf{y}_t = \mathbf{S}_t^\top \phi(\mathbf{q}_t), \qquad \mathbf{S}_t = (1 - \eta_t\lambda_t)\mathbf{S}_{t-1} + \eta_t\, \mathbf{x}_t\mathbf{v}_t^\top, \qquad \mathbf{x}_t := \phi(\mathbf{k}_{t-1}).$$

If one uses the normalized variant, the matched numerator/normalizer recurrence is

$$\mathbf{y}_t = \frac{\mathbf{S}_t^\top \phi(\mathbf{q}_t)}{\mathbf{z}_t^\top \phi(\mathbf{q}_t) + \varepsilon_{\mathrm{attn}}}, \qquad \mathbf{S}_t = \gamma_t\mathbf{S}_{t-1} + \eta_t\, \mathbf{x}_t\mathbf{v}_t^\top, \qquad \mathbf{z}_t = \gamma_t\mathbf{z}_{t-1} + \eta_t\, \mathbf{x}_t, \qquad \gamma_t := 1 - \eta_t\lambda_t.$$

With $\mathbf{z}_0 \geq 0$ and elementwise nonnegative feature maps, this normalized form remains attention-like only when $\gamma_t \geq 0$ for all t; otherwise $\mathbf{z}_t$ can change sign even if $\phi(\cdot) \geq 0$. Accordingly, whenever we use the normalized read we enforce the positive-decay clamp (equivalently $\eta_t\lambda_t \leq 1 - \varepsilon_\gamma$), or else use the denominator-free form.

The additive normalized case in Eq. (2.4) is recovered by $\eta_t \equiv 1$ and $\lambda_t \equiv 0$.

Log-space chunk-local renormalization (ctxλ). In long-context settings, direct products of the decay factors can become numerically fragile in mixed precision. This reparameterization is used only when the decay is positive. Define

$$\alpha_t := \eta_t\lambda_t, \qquad \gamma_t := 1 - \alpha_t,$$

and, whenever positivity is not guaranteed a priori, clamp $\alpha_t \leftarrow \min(\alpha_t, 1 - \varepsilon_\gamma)$ before computing

$$\log \gamma_t := \mathrm{log1p}(-\alpha_t)$$

in fp32. Over a chunk of timesteps $t \in \{a, \ldots, b\}$, we accumulate multiplicative decay via sums of $\log \gamma_t$, i.e.

$$\prod_{i=a}^{t} \gamma_i \;=\; \exp\Big(\sum_{i=a}^{t} \log \gamma_i\Big),$$

and apply the corresponding rescaling to both the numerator state (e.g., $\mathbf{S}_t$) and the normalizer state (e.g., $\mathbf{z}_t$) within the chunk. For the normalized readout

$$\mathbf{y}_t = \frac{\mathbf{S}_t^\top \phi(\mathbf{q}_t)}{\mathbf{z}_t^\top \phi(\mathbf{q}_t) + \varepsilon_{\mathrm{attn}}},$$

exact output preservation additionally requires forming the ratio after undoing the common rescaling, or equivalently, rescaling the stabilizer by the same factor. With a fixed $\varepsilon_{\text{attn}} > 0$ and directly using the renormalized states inside the ratio, the equivalence is only approximate. The denominator-free recurrence is unaffected by this caveat.

Scaled DeltaNet. For regression-style fast weights, we use the same scaled write-feature $\mathbf{x}_t$ inside the NLMS update:

$$\mathbf{S}_t = (1 - \eta_t \lambda_t)\mathbf{S}_{t-1} + \eta_t\, \mathbf{x}_t(\mathbf{v}_t - \mathbf{S}_{t-1}^\top \mathbf{x}_t)^\top, \qquad \eta_t = \frac{\beta_t}{\|\mathbf{x}_t\|_2^2 + \lambda_t + \varepsilon}.$$

To keep λ_{scale} comparable between scaled and unscaled implementations, we parameterize a dimensionless base ridge

$$\bar{\lambda}_t := \lambda_{\text{scale}}\, \sigma(\widetilde{\lambda}_t), \qquad \lambda_t^{\text{eff}} := \bar{\lambda}_t\, \|\mathbf{x}_t\|_2^2,$$

so in every non-sliding scaled variant the network-produced quantity is the *dimensionless* ratio $\bar{\lambda}_t = \lambda_t^{\text{eff}}/\|\mathbf{x}_t\|_2^2$ rather than an absolute ridge coefficient. The actual ridge that enters the recurrence is λ_t^{eff}. We then use λ_t^{eff} in both the NLMS denominator and decay path:

$$\eta_t = \frac{\beta_t}{\|\mathbf{x}_t\|_2^2 + \lambda_t^{\text{eff}} + \varepsilon}, \qquad \gamma_t = 1 - \eta_t \lambda_t^{\text{eff}}.$$

Hence, for $\|\mathbf{x}_t\|_2 > 0$,

$$\eta_t \lambda_t^{\text{eff}} = \frac{\beta_t \bar{\lambda}_t}{1 + \bar{\lambda}_t + \varepsilon/\|\mathbf{x}_t\|_2^2},$$

and, when $\varepsilon = 0$ and $\|\mathbf{x}_t\|_2 > 0$,

$$\eta_t \lambda_t^{\text{eff}} = \frac{\beta_t \bar{\lambda}_t}{1 + \bar{\lambda}_t}, \qquad \eta_t\, \mathbf{x}_t \mathbf{x}_t^\top = \frac{\beta_t}{1 + \bar{\lambda}_t}\, \frac{\mathbf{x}_t \mathbf{x}_t^\top}{\|\mathbf{x}_t\|_2^2}.$$

Thus, when $\varepsilon = 0$ and $\|\mathbf{x}_t\|_2 > 0$, both the scalar decay fraction and the projector term are exactly invariant to a uniform rescaling $\mathbf{x}_t \mapsto c\,\mathbf{x}_t$; for $\varepsilon > 0$, this invariance is only approximate and becomes exact in the usual regime $\varepsilon \ll \|\mathbf{x}_t\|_2^2$. When $\|\mathbf{x}_t\|_2 = 0$ we use the no-op convention discussed earlier; the recurrence remains well-defined, but these ratio identities are not invoked. This is the precise scale-robustness property used by the update. It does *not* imply that the entire block is literally unchanged unless the read path is rescaled consistently as well. For sliding-window variants, replace $\|\mathbf{x}_t\|_2^2$ with the window-average write energy $\bar{E}_t^{(B)}$. Equivalently, for sliding-window variants, we use

$$\bar{E}_t^{(B)} := \frac{1}{B_t} \sum_{j \in \mathcal{I}_t} \|\mathbf{x}_j\|_2^2, \qquad \lambda_t^{\text{eff},(B)} := \bar{\lambda}_t\, \bar{E}_t^{(B)}.$$

Thus, the same interpretation carries over: the learned/base quantity $\bar{\lambda}_t$ is dimensionless, while the actual sliding-window ridge is $\lambda_t^{\text{eff},(B)}$. Whenever this scale-coupled parameterization is active, the actual coefficient appearing in the fast-memory objective, smoothness constant, decay fraction, and descent statements is λ_t^{eff} (or $\lambda_t^{\text{eff},(B)}$ in the sliding-window case), whereas $\bar{\lambda}_t$ is only the bounded base parameter produced by the network. In the current sliding-state implementations, $\bar{E}_t^{(B)}$ is treated as a statistics-only multiplier when forming $\lambda_t^{\text{eff},(B)}$ (equivalently, the multiplier is detached / stop-gradient), while the NLMS denominator still uses the live $\bar{E}_t^{(B)}$.

Scope of the scale-coupled ridge parameterization. The discussion above applies to the *regression-style* updates (FALCON-2 and FALCON-3), where the feature energy enters the smoothness bound, and the ridge term genuinely belongs to the local objective. It should not be carried over unchanged to the sliding inner-product update FALCON-3A. In FALCON-3A, the feature-energy term is used only to normalize the write magnitude, and the recurrence is controlled more naturally by a write coefficient and a direct decay fraction (Section 4.5); parameterizing a raw ridge there unnecessarily couples forgetting to the current write amplitude.

Ctxλ reduction via log-decay and local renormalization. For scalar-decay recurrences of the affine form

$$\mathbf{S}_t = (\gamma_t \mathbf{I} - \eta_t \mathbf{x}_t \mathbf{x}_t^\top)\mathbf{S}_{t-1} \;+\; \eta_t\, \mathbf{x}_t \mathbf{y}_t^\top, \qquad \gamma_t > 0,$$

with $\mathbf{y}_t$ the write target (e.g. $\mathbf{v}_t$). Here γ_t denotes the scalar carry: in the unclamped recurrence $\gamma_t = 1 - \eta_t \lambda_t$, while with the implementation clamp we use $\gamma_t := 1 - \alpha_t$ with $\alpha_t = \min(\eta_t \lambda_t, 1 - \varepsilon_\gamma)$. Define

$$c_0 := 1, \qquad c_t := \prod_{r=1}^{t} \gamma_r, \qquad \widetilde{\mathbf{S}}_t := \mathbf{S}_t / c_t.$$

Then

$$\widetilde{\mathbf{S}}_t = (\mathbf{I} - \widetilde{\eta}_t\, \mathbf{x}_t \mathbf{x}_t^\top)\widetilde{\mathbf{S}}_{t-1} \;+\; \widetilde{\eta}_t\, \mathbf{x}_t \widetilde{\mathbf{y}}_t^\top, \qquad \widetilde{\eta}_t := \frac{\eta_t}{\gamma_t}, \qquad \widetilde{\mathbf{y}}_t := \frac{\mathbf{y}_t}{c_{t-1}}.$$

Thus, exact equivalence requires *both* step-size rescaling and inverse-decay rescaling of the write target; rescaling η_t alone is not sufficient. In chunk-parallel implementations, we do not form global products c_t; instead, we accumulate $\log \gamma_t$ within each chunk and apply the equivalent local renormalization to boundary states and write targets.

4.2 Regression Loss (Delta Network)

We formulate the state update as an autoregressive linear regression problem. At time step t, let the state be $\mathbf{S}_{t-1} \in \mathbb{R}^{d_x \times d_v}$, the write feature be $\mathbf{x}_t \triangleq \phi(\mathbf{k}_{t-1}) \in \mathbb{R}^{d_x}$, and the target be $\mathbf{y}_t \triangleq \mathbf{v}_t \in \mathbb{R}^{d_v}$. (In the unkernelized case, ϕ is the identity and $d_x = d$.) The state $\mathbf{S}_{t-1}$ acts as a linear predictor mapping the prefix feature $\mathbf{x}_t$ to the newly observed target $\mathbf{y}_t$.

The instantaneous squared-error loss is defined as:

$$f_t(\mathbf{S}_{t-1}) = \frac{1}{2} \|\mathbf{S}_{t-1}^\top \mathbf{x}_t - \mathbf{y}_t\|_2^2. \tag{4.1}$$

Let $\mathbf{r}_t \triangleq \mathbf{y}_t - \mathbf{S}_{t-1}^\top \mathbf{x}_t$ denote the residual. The gradient with respect to the state is:

$$\nabla_{\mathbf{S}} f_t(\mathbf{S}_{t-1}) = \mathbf{x}_t (\mathbf{S}_{t-1}^\top \mathbf{x}_t - \mathbf{y}_t)^\top = -\,\mathbf{x}_t \mathbf{r}_t^\top. \tag{4.2}$$

Delta update (standard vs. next-latent). A single step of Online Gradient Descent (OGD) with learning rate η_t yields the Delta update:

$$\begin{aligned} \mathbf{S}_t &= \mathbf{S}_{t-1} - \eta_t \nabla_{\mathbf{S}} f_t(\mathbf{S}_{t-1}) \\ &= \mathbf{S}_{t-1} + \eta_t \mathbf{x}_t \mathbf{r}_t^\top \\ &= (\mathbf{I}_{d_x} - \eta_t \mathbf{x}_t \mathbf{x}_t^\top)\mathbf{S}_{t-1} + \eta_t \mathbf{x}_t \mathbf{y}_t^\top. \end{aligned} \tag{4.3}$$

Under next-latent alignment we have $\mathbf{x}_t = \phi(\mathbf{k}_{t-1})$ and $\mathbf{y}_t = \mathbf{v}_t$. Replacing $\mathbf{k}_{t-1}$ with $\mathbf{k}_t$ (equivalently, $\mathbf{x}_t \leftarrow \phi(\mathbf{k}_t)$) recovers the unshifted DeltaNet update of Schlag et al. (2021).

L_2 Regularization. Adding an L_2 penalty $\frac{\lambda_t}{2}\|\mathbf{S}\|_F^2$ to the instantaneous loss yields the shrinkage term $-\eta_t\lambda_t\mathbf{S}_{t-1}$ in the online update (cf. Eq. (3.3)). Under normalized step sizes, this results in the multiplicative factor $(1-\eta_t\lambda_t)$, which we treat as a simple and controllable forgetting mechanism.

FALCON-2: Adaptive Learning Rates. We propose **FALCON-2**, which combines NLMS-style normalization with per-channel adaptive learning rates. In Appendix D, we derive a vectorized dual form that makes column-wise adaptivity computationally tractable on GPUs. This allows the step size to be a vector $\boldsymbol{\eta}_t \in \mathbb{R}^{d_v}$, tailoring the update magnitude for each value channel (column of $\mathbf{S}$) independently.

Per-column NLMS step sizes. Concretely, we use an NLMS-style normalizer shared across columns and learn per-column gains (equivalently, per output-feature / column gains):

$$\eta_{j,t} = \frac{\beta_{j,t}}{\|\mathbf{x}_t\|_2^2 + \lambda_t + \varepsilon}, \qquad \beta_{j,t} \in (0,2),\ \varepsilon > 0.$$

When $\lambda_t = 0$, $\varepsilon = 0$, and the gains are tied across channels ($\beta_{j,t} \equiv \beta_t$), this reduces to the classical scalar NLMS step size; otherwise it is a column-wise NLMS generalization with a shared normalizer. Since the squared-error (ridge) loss decomposes across value coordinates (columns of $\mathbf{S}$), the per-step descent argument from Lemma 3.1 applies column-wise provided $0 < \beta_{j,t} < 2$ for all j.

Update rule. Let $\boldsymbol{\eta}_t = (\eta_{1,t}, \ldots, \eta_{d_v,t})^\top$. The update can be written compactly as:

$$\mathbf{S}_t = \mathbf{S}_{t-1}\Big(\mathbf{I}_{d_v} - \lambda_t\ \mathrm{Diag}(\boldsymbol{\eta}_t)\Big) + \mathbf{x}_t(\boldsymbol{\eta}_t \odot \mathbf{r}_t)^\top, \tag{4.4}$$

Expanding $\mathbf{r}_t = \mathbf{y}_t - \mathbf{S}_{t-1}^\top\mathbf{x}_t$ makes the regression-induced targeted edit explicit:

$$\mathbf{S}_t = \mathbf{S}_{t-1}\Big(\mathbf{I}_{d_v} - \lambda_t\ \mathrm{Diag}(\boldsymbol{\eta}_t)\Big) - \mathbf{x}_t\Big(\mathbf{x}_t^\top\mathbf{S}_{t-1}\,\mathrm{Diag}(\boldsymbol{\eta}_t)\Big) + \mathbf{x}_t(\boldsymbol{\eta}_t \odot \mathbf{y}_t)^\top.$$

This highlights the two edit/shrinkage mechanisms in FALCON-2: per-column right-multiplicative shrinkage (the weight-decay path) and a per-column rank-one edit along the current write-feature direction.

Relation to RWKV-7 and Kimi Linear Attention. RWKV-style recurrent LLMs (Peng et al., 2023) and more recent delta-rule variants (e.g., RWKV-7 and Kimi Linear Attention (Team et al., 2025)) enrich Delta-style fast weights with learned gating and in-context learning-rate control. Our formulation is complementary: we ground the write rule in a causal next-latent regression objective, make the required one-step key shift explicit, and stabilize learning with NLMS normalization. The same normalization/alignment can be used as a drop-in replacement for the local update inside delta-style blocks, independent of their particular gating or mixing parameterization.

Equivalently, writing the update column-wise, with $\mathbf{s}_{t,j}$ the j-th column of $\mathbf{S}_t$ and $(\mathbf{y}_t)_j$ the j-th coordinate of $\mathbf{y}_t$,

$$\mathbf{s}_{t,j} = (1 - \lambda_t\eta_{j,t})\,\mathbf{s}_{t-1,j} + \eta_{j,t}\,\mathbf{x}_t\,((\mathbf{y}_t)_j - \langle \mathbf{x}_t, \mathbf{s}_{t-1,j}\rangle). \tag{4.5}$$

This formulation allows the model to learn a per-column update velocity via $\boldsymbol{\eta}_t$, while the ridge path contributes the per-column shrinkage factor $(1 - \lambda_t\eta_{j,t})$.

Log-space forgetting (ctxλ) and exact column-wise reduction. For the unclamped update, the per-column shrinkage factor is

$$\gamma_{j,t}^{\mathrm{raw}} := 1 - \lambda_t \eta_{j,t}.$$

Because the admissible gains allow $\lambda_t \eta_{j,t} > 1$, one can have $\gamma_{j,t}^{\mathrm{raw}} < 0$; the log-space reduction therefore requires a positive-decay implementation. Whenever we unroll in log space, we use

$$\alpha_{j,t}^{\mathrm{raw}} := \lambda_t \eta_{j,t}, \qquad \alpha_{j,t} := \min(\alpha_{j,t}^{\mathrm{raw}},\, 1 - \varepsilon_\gamma), \qquad \gamma_{j,t} := 1 - \alpha_{j,t} \in (0, 1],$$

and compute $\log \gamma_{j,t} = \mathrm{log1p}(-\alpha_{j,t})$ in fp32. Define

$$c_{0,j} := 1, \qquad c_{t,j} := \prod_{s=1}^{t} \gamma_{s,j}, \qquad \widetilde{\mathbf{s}}_{t,j} := \mathbf{s}_{t,j} / c_{t,j}.$$

Then, the implemented positive-decay column-wise recurrence is equivalent to the projector-only form

$$\widetilde{\mathbf{s}}_{t,j} = (\mathbf{I} - \widetilde{\eta}_{j,t} \mathbf{x}_t \mathbf{x}_t^\top) \widetilde{\mathbf{s}}_{t-1,j} \;+\; \widetilde{\eta}_{j,t}\, \widetilde{v}_{t,j}\, \mathbf{x}_t, \qquad \widetilde{\eta}_{j,t} := \frac{\eta_{j,t}}{\gamma_{j,t}}, \qquad \widetilde{v}_{t,j} := \frac{v_{t,j}}{c_{t-1,j}}.$$

Hence, exact equivalence requires both the step-size rescaling $\widetilde{\eta}_{j,t} = \eta_{j,t}/\gamma_{j,t}$ and inverse-decay rescaling of the per-column write target. When the clamp is inactive ($\lambda_t \eta_{j,t} < 1 - \varepsilon_\gamma$), this coincides exactly with the unclamped recurrence; otherwise, it is the numerically safe surrogate with smaller effective decay in the shrinkage path. In practice, we implement this with chunk-local log-prefix decays rather than global products $c_{t,j}$; Appendix D gives the explicit kernel.

Chunk-parallel implementation. FALCON-2 can be trained sequence-parallel by chunking the length axis and using a Gram/WY representation within each chunk. With per-channel step sizes, each value dimension induces its own unit-lower-triangular system, but all systems share the same base Gram matrix $\mathbf{G} = \mathbf{K}^\top \mathbf{K}$ inside a chunk. Algorithm 2 summarizes a chunk-wise forward pass for the projector-only case ($\lambda_t = 0$); Appendix D derives this dual form and discusses the $\lambda_t > 0$ extension.

4.3 Inner Product Loss (Linear Attention and Mamba-2)

We now consider an Inner Product objective that encourages alignment between the state prediction and the target. We write it in minimization form and optionally add an L_2 penalty:

$$\ell_t^{\mathrm{ip}}(\mathbf{S}) \triangleq -\langle \mathbf{S}^\top \mathbf{x}_t, \mathbf{y}_t \rangle + \frac{\lambda_t}{2} \|\mathbf{S}\|_F^2, \qquad \lambda_t \geq 0. \tag{4.6}$$

When $\lambda_t = 0$, the objective is linear in $\mathbf{S}$ (no finite minimizer) and gradient descent reduces to purely additive Hebbian writes.

Standard vs. next-latent alignment. If we choose the unshifted features $(\mathbf{x}_t, \mathbf{y}_t) = (\phi(\mathbf{k}_t), \mathbf{v}_t)$ (or $(\mathbf{k}_t, \mathbf{v}_t)$ in the unkernelized case), the additive update below matches the usual Linear Attention write $\phi(\mathbf{k}_t)\mathbf{v}_t^\top$ (Eq. (2.3)). Our next-latent framework instead uses $(\mathbf{x}_t, \mathbf{y}_t) = (\phi(\mathbf{k}_{t-1}), \mathbf{v}_t)$, yielding a one-step shifted write stream.

FALCON-A variants. We refer to inner-product fast-weight instantiations as FALCON-2A for the non-sliding update (Eq. (2.5)) and FALCON-3A for the sliding-window update in Section 4.5.

Algorithm 2 Parallelized FALCON-2 (Chunk-wise Forward, $\lambda_t = 0$)

Require: Shifted write features $\mathbf{K} \in \mathbb{R}^{L\times d_x}$, query features $\mathbf{Q} \in \mathbb{R}^{L\times d_x}$, values $\mathbf{V} \in \mathbb{R}^{L\times d_v}$, per-channel step sizes $\boldsymbol{\eta} \in \mathbb{R}^{L\times d_v}$ with $\eta_{t,j} \geq 0$. Chunk size C (assume $C \mid L$).

Ensure: Outputs $\mathbf{O} \in \mathbb{R}^{L\times d_v}$ for the projector-only recurrence ($\lambda_t = 0$).

Partition the length-L axis into $M = L/C$ contiguous chunks. For each chunk m, let $\mathbf{K}_{(m)}, \mathbf{Q}_{(m)} \in \mathbb{R}^{C\times d_x}$ and $\mathbf{V}_{(m)} \in \mathbb{R}^{C\times d_v}$ and $\boldsymbol{\eta}_{(m)} \in \mathbb{R}^{C\times d_v}$ denote the usual row-major slices.

For the WY/Gram algebra, use column-major chunk matrices

$$\mathbf{K}^{(m)} := \mathbf{K}_{(m)}^\top \in \mathbb{R}^{d_x\times C}, \qquad \mathbf{Q}^{(m)} := \mathbf{Q}_{(m)}^\top \in \mathbb{R}^{d_x\times C}.$$

Initialize $\mathbf{S}_{\text{in}} \leftarrow \mathbf{0}_{d_x\times d_v}$ ▷ (Or a provided initial state)

Initialize output buffer $\mathbf{O} \leftarrow \mathbf{0}_{L\times d_v}$.

for $m = 1, \ldots, M$ **do**

  $\mathbf{K} \leftarrow \mathbf{K}^{(m)}$, $\mathbf{Q} \leftarrow \mathbf{Q}^{(m)}$, $\mathbf{V} \leftarrow \mathbf{V}_{(m)}$, $\boldsymbol{\eta} \leftarrow \boldsymbol{\eta}_{(m)}$

  $\boldsymbol{\Sigma} \leftarrow \sqrt{\boldsymbol{\eta}}$ ▷ $C \times d_v$

  $\mathbf{G} \leftarrow \mathbf{K}^\top\mathbf{K}$ ▷ $C \times C$ (shared base Gram)

  $\mathbf{M} \leftarrow \mathrm{tril}(\mathbf{Q}^\top\mathbf{K}, 0)$ ▷ $C \times C$ (unscaled scores; read-after-write)

  $\mathbf{H} \leftarrow \mathbf{Q}^\top\mathbf{S}_{\text{in}}, \quad \mathbf{P} \leftarrow \mathbf{K}^\top\mathbf{S}_{\text{in}}$ ▷ $C \times d_v$

  ▷ For each channel j, build $\mathbf{L}_j = \mathbf{I} + \mathrm{tril}(\mathbf{G} \odot (\boldsymbol{\Sigma}_{:,j}\boldsymbol{\Sigma}_{:,j}^\top), -1)$ and solve $\mathbf{L}_j\mathbf{z}_j = \boldsymbol{\Sigma}_{:,j} \odot \mathbf{V}_{:,j}$ and $\mathbf{L}_j\boldsymbol{\gamma}_j = \boldsymbol{\Sigma}_{:,j} \odot \mathbf{P}_{:,j}$

  $\mathbf{Z} \leftarrow \mathrm{BatchedSolveTri}(\{\mathbf{L}_j\}_{j=1}^{d_v}, \mathbf{V} \odot \boldsymbol{\Sigma})$ ▷ $C \times d_v$; column j is $\mathbf{z}_j$

  $\boldsymbol{\Gamma} \leftarrow \mathrm{BatchedSolveTri}(\{\mathbf{L}_j\}_{j=1}^{d_v}, \mathbf{P} \odot \boldsymbol{\Sigma})$ ▷ $C \times d_v$; column j is $\boldsymbol{\gamma}_j$

  $\mathbf{B} \leftarrow \boldsymbol{\Sigma} \odot (\mathbf{Z} - \boldsymbol{\Gamma})$ ▷ $C \times d_v$

  $\mathbf{O}^{(m)} \leftarrow \mathbf{H} + \mathbf{MB}$ ▷ $C \times d_v$ (chunk outputs)

  $\mathbf{S}_{\text{out}} \leftarrow \mathbf{S}_{\text{in}} + \mathbf{KB}$ ▷ $d_x \times d_v$ (chunk-exit state)

  Write $\mathbf{O}_{(m)} \leftarrow \mathbf{O}^{(m)}$ ▷ store into rows $(m-1)C+1 : mC$ of $\mathbf{O}$

  $\mathbf{S}_{\text{in}} \leftarrow \mathbf{S}_{\text{out}}$ ▷ propagate to next chunk

end for

return $\mathbf{O}$.

Gradient and update. The gradient of Eq. (4.6) is

$$\nabla_{\mathbf{S}}\ell_t^{\text{ip}}(\mathbf{S}) = -\mathbf{x}_t\mathbf{y}_t^\top + \lambda_t\mathbf{S}.$$

A gradient step with rate η_t yields the Linear-Attention/Mamba-2 style update

$$\mathbf{S}_t = (1 - \eta_t\lambda_t)\mathbf{S}_{t-1} + \eta_t\,\mathbf{x}_t\mathbf{y}_t^\top, \tag{4.7}$$

where $\mathbf{x}_t = \phi(\mathbf{k}_{t-1})$ and $\mathbf{y}_t = \mathbf{v}_t$ under next-latent alignment. Setting $\lambda_t = 0$ recovers the usual additive write $\mathbf{S}_t = \mathbf{S}_{t-1} + \eta_t\mathbf{x}_t\mathbf{y}_t^\top$.

FALCON-2A step size. For $\lambda_t > 0$, ℓ_t^{ip} is λ_t-smooth (its Hessian is $\lambda_t\mathbf{I}$), so any $\eta_t \in (0, 2/\lambda_t)$ yields per-step descent; importantly, this descent condition depends only on λ_t (the curvature comes solely from the ridge term), not on $\|\mathbf{x}_t\|_2$. In practice, we *also* normalize by the write-feature energy to control the magnitude of the additive write $\mathbf{x}_t\mathbf{y}_t^\top$ when feature norms vary (e.g., under kernel maps)

and to obtain a well-behaved $\lambda_t \to 0$ limit:

$$\eta_t = \frac{\beta_t}{\|\mathbf{x}_t\|_2^2 + \lambda_t + \varepsilon}, \qquad \beta_t \in (0, 2),\ \varepsilon \geq 0. \tag{4.8}$$

As in Eq. (3.4), we set $\eta_t := 0$ when the denominator vanishes. When $\lambda_t > 0$, this choice satisfies $\eta_t \leq \beta_t/\lambda_t$ (tight when $\|\mathbf{x}_t\|_2 = 0$ and $\varepsilon = 0$), and hence $\eta_t < 2/\lambda_t$ for any $\beta_t \in (0, 2)$. Since ℓ_t^{ip} is λ_t-smooth (its Hessian is $\lambda_t \mathbf{I}$), Lemma 3.1 yields per-step descent in ℓ_t^{ip} for this step size. The $\|\mathbf{x}_t\|_2^2$ term is not required by curvature, but stabilizes the write magnitude and yields a sensible $\lambda_t \to 0$ limit. When $\lambda_t = 0$, ℓ_t^{ip} has no finite minimizer; the same normalization (with $\lambda_t = 0$) should be interpreted purely as a magnitude stabilizer (take $\varepsilon > 0$, or set $\eta_t = 0$ when the denominator vanishes).

Decay positivity. Some parallel/unrolled forms (Section 4.6) use $\gamma_t := 1 - \eta_t \lambda_t$ in log space and therefore require $\gamma_t > 0$. In implementations, compute $\alpha_t := \eta_t \lambda_t$ and, if necessary, clamp $\alpha_t \leftarrow \min(\alpha_t, 1 - \varepsilon_\gamma)$ before computing $\log \gamma_t = \operatorname{log1p}(-\alpha_t)$ (fp32). The clamp is inactive whenever $\eta_t \lambda_t < 1 - \varepsilon_\gamma$; in that regime, the dynamics match $\gamma_t = 1 - \eta_t \lambda_t$ exactly. As above, the descent statement pertains to the unclamped recurrence; when the clamp activates, the update should be viewed as a numerically safe surrogate with a smaller effective decay in the shrinkage path.

4.4 Mini-batch Update Rule for Regression (FALCON-3)

To better capture local dependencies and reduce noise accumulation, we introduce a *sliding-state* mechanism: instead of updating the state from only the instantaneous residual, we take a single mini-batch gradient step on a finite history window of nominal size B.

Relation to ATLAS Omega rule. ATLAS (Behrouz et al., 2025a) introduces the Omega learning rule, which updates a memory module by performing gradient-based optimization of an internal objective over a sliding context window, rather than optimizing only on the most recent token. FALCON-3 can be viewed as a specialization of this sliding-window principle to (i) a linear matrix memory $\mathbf{S}$, (ii) a squared-error regression objective, and (iii) uniform window weights. Our formulation additionally enforces strict autoregressive causality through next-latent alignment (writing with $(\mathbf{k}_{t-1}, \mathbf{v}_t)$ under a read-after-write ordering) and uses a conservative trace-normalized step size, enabling stable training while preserving compatibility with chunk-parallel scan implementations.

Exact streaming state. For FALCON-3 and FALCON-3A, the matrix state $\mathbf{S}_t$ alone is *not* Markov for exact continuation: the next update also depends on the active window $\mathcal{I}_t$. Exact continuation across a segment boundary therefore requires, in addition to $\mathbf{S}_t$, a fixed-width boundary tail containing the last $B-1$ causal pairs $\{(\mathbf{x}_j, \mathbf{v}_j)\}_{j=\max(2,t-B+2)}^{t}$ (or an equivalent rolling-window representation). Resetting this tail at a segment boundary changes the update near the boundary and should be viewed as an explicit boundary reset rather than an exact continuation of the same recurrence.

Sequence-parallel training. FALCON-3 is a rank-at-most-B low-rank Delta rule (Omega rule). After zero-padding each active window to width B, the recurrence has a fixed rank-B driver at every step. Algorithm 3 gives the reference sequential update. For hardware-efficient training, we realize the same recurrence chunk-parallel with ParallelFlow's triangular-tensor solver (`tensorInv`), using the exact positive-decay reduction in Eq. (4.15). Algorithm 4 gives explicit pseudocode for this implementation. Inside `tensorInv`, we use a *block*-strict-causal mask over (time, rank) pairs, so

Algorithm 3 FALCON-3 (recurrent form)

Require: Query features $\{\mathbf{q}_t\}_{t=1}^T$ with $\mathbf{q}_t = \phi(\mathbf{q}_t)$, shifted write-features $\{\mathbf{x}_t\}_{t=1}^T$ with $\mathbf{x}_1 = \mathbf{0}$ and $\mathbf{x}_t = \phi(\mathbf{k}_{t-1})$ for $t \geq 2$, values $\{\mathbf{v}_t\}_{t=1}^T$, window size B, gains $\beta_t \in (0, 2)$, ridge $\lambda_t \geq 0$, stabilizer $\varepsilon > 0$, init $\mathbf{S}_0 = \mathbf{0}$.

Ensure: Outputs $\mathbf{O} \in \mathbb{R}^{T \times d_v}$ with row t equal to $\mathbf{o}_t^\top = \mathbf{q}_t^\top \mathbf{S}_t$ (read-after-write), final matrix state $\mathbf{S}_T$, and final FIFO buffer $\mathcal{W}_T$; exact continuation across a later segment requires both $\mathbf{S}_T$ and $\mathcal{W}_T$.

Initialize empty FIFO buffer $\mathcal{W} \leftarrow [\,]$ and output buffer $\mathbf{O} \leftarrow \mathbf{0}$.
for $t = 1, \ldots, T$ **do**
  if $t = 1$ **then**
    $\eta_t \leftarrow 0, \quad \mathbf{S}_t \leftarrow \mathbf{S}_{t-1}, \quad \mathbf{o}_t \leftarrow \mathbf{S}_t^\top \mathbf{q}_t$
    Write $\mathbf{o}_t^\top$ into row t of $\mathbf{O}$ and **continue**
  end if
  Push $(\mathbf{x}_t, \mathbf{v}_t)$ into $\mathcal{W}$; if $|\mathcal{W}| > B$, pop the oldest pair.
  $B_t \leftarrow |\mathcal{W}|, \quad \mathbf{U}_t \leftarrow \mathbf{0}_{d_x \times d_v}, \quad E_t \leftarrow 0$
  for all $(\mathbf{x}, \mathbf{v}) \in \mathcal{W}$ **do**
    $\mathbf{U}_t \leftarrow \mathbf{U}_t + \mathbf{x}\big(\mathbf{v} - \mathbf{S}_{t-1}^\top \mathbf{x}\big)^\top$
    $E_t \leftarrow E_t + \|\mathbf{x}\|_2^2$
  end for
  $\bar{E}_t^{(B)} \leftarrow E_t / B_t$
  $\eta_t \leftarrow \beta_t / (\bar{E}_t^{(B)} + \lambda_t + \varepsilon)$
  $\mathbf{S}_t \leftarrow (1 - \eta_t \lambda_t)\mathbf{S}_{t-1} + (\eta_t / B_t)\, \mathbf{U}_t$
  $\mathbf{o}_t \leftarrow \mathbf{S}_t^\top \mathbf{q}_t$
  Write $\mathbf{o}_t^\top$ into row t of $\mathbf{O}$
end for
return $(\mathbf{O}, \mathbf{S}_T, \mathcal{W})$

same-time rank components do not interact and every residual is evaluated at the pre-update state $\mathbf{S}_{t-1}$. Output materialization then uses an inclusive block-causal mask to implement read-after-write. Appendix F derives the driver construction. The inner-product counterpart (FALCON-3A) admits an explicit masked linear-attention form (Section 4.6), enabling fully vectorized training over the sequence dimension.

For $t \geq 2$, let the active window indices be $\mathcal{I}_t = \{j \mid \max(2, t - B + 1) \leq j \leq t\}$ and denote the realized window size by $B_t := |\mathcal{I}_t| \leq B$ (so $B_t \geq 1$). Define the write feature $\mathbf{x}_j := \phi(\mathbf{k}_{j-1})$ (so $\mathbf{x}_j = \mathbf{k}_{j-1}$ when ϕ is the identity). We use the boundary convention $\mathbf{x}_1 = \phi(\mathbf{k}_0) = \mathbf{0}$ and set $\eta_1 := 0$, so the $t = 1$ write is a no-op; all windowed objectives/updates below are defined for $t \geq 2$. To make the update magnitude (and hence the effective decay) invariant to the nominal window size B, we optimize the *window-average* squared loss:

$$\ell_t^{\mathrm{reg,(B)}}(\mathbf{S}_{t-1}) = \frac{1}{2B_t} \sum_{j \in \mathcal{I}_t} \|\mathbf{S}_{t-1}^\top \mathbf{x}_j - \mathbf{v}_j\|_2^2 + \frac{\lambda_t}{2} \|\mathbf{S}_{t-1}\|_F^2, \qquad (t \geq 2). \tag{4.9}$$

Sufficient Statistics. We define the sliding covariance $\mathbf{C}_t^{(B)}$ and cross-covariance $\mathbf{N}_t^{(B)}$ matrices:

$$\mathbf{C}_t^{(B)} \triangleq \sum_{j \in \mathcal{I}_t} \mathbf{x}_j \mathbf{x}_j^\top, \qquad \mathbf{N}_t^{(B)} \triangleq \sum_{j \in \mathcal{I}_t} \mathbf{x}_j \mathbf{v}_j^\top. \tag{4.10}$$

Define the window-averaged statistics

$$\bar{\mathbf{C}}_t^{(B)} := \frac{1}{B_t} \mathbf{C}_t^{(B)}, \qquad \bar{\mathbf{N}}_t^{(B)} := \frac{1}{B_t} \mathbf{N}_t^{(B)}.$$

Then the gradient of the window-average loss is

$$\nabla_{\mathbf{S}}\ell_t^{\mathrm{reg,(B)}}(\mathbf{S}_{t-1}) = \bar{\mathbf{C}}_t^{(B)}\mathbf{S}_{t-1} - \bar{\mathbf{N}}_t^{(B)} + \lambda_t\mathbf{S}_{t-1}.$$

Update Rule. We apply a block-normalized gradient step:

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \eta_t\nabla_{\mathbf{S}}\ell_t^{\mathrm{reg,(B)}}(\mathbf{S}_{t-1}). \tag{4.11}$$

Substituting the gradient yields the affine update

$$\mathbf{S}_t = (\mathbf{I}_{d_x} - \eta_t(\bar{\mathbf{C}}_t^{(B)} + \lambda_t\mathbf{I}_{d_x}))\mathbf{S}_{t-1} + \eta_t\bar{\mathbf{N}}_t^{(B)}. \tag{4.12}$$

Equivalently, collecting residuals at the pre-update state yields

$$\mathbf{S}_t = (1-\eta_t\lambda_t)\mathbf{S}_{t-1} \;+\; \frac{\eta_t}{B_t}\sum_{j\in\mathcal{I}_t}\mathbf{x}_j\big(\mathbf{v}_j - \mathbf{S}_{t-1}^\top\mathbf{x}_j\big)^\top. \tag{4.13}$$

This is the direct mini-batch analogue of Eq. (3.3): all window residuals are evaluated at the pre-update state $\mathbf{S}_{t-1}$, and the update averages their rank-one gradients.

For $t \geq 2$, we use the conservative trace/energy normalization on the *window-average* covariance:

$$\bar{E}_t^{(B)} := \mathrm{tr}(\bar{\mathbf{C}}_t^{(B)}) = \frac{1}{B_t}\sum_{j\in\mathcal{I}_t}\|\mathbf{x}_j\|_2^2 = \frac{\mathrm{tr}(\mathbf{C}_t^{(B)})}{B_t}, \qquad \eta_t = \frac{\beta_t}{\bar{E}_t^{(B)} + \lambda_t + \varepsilon}, \qquad \beta_t \in (0,2),\ \varepsilon > 0.$$

Since $\ell_t^{\mathrm{reg,(B)}}$ is $L_t^{(B)}$-smooth with $L_t^{(B)} = \lambda_{\max}(\bar{\mathbf{C}}_t^{(B)}) + \lambda_t$ and $\lambda_{\max}(\bar{\mathbf{C}}_t^{(B)}) \leq \mathrm{tr}(\bar{\mathbf{C}}_t^{(B)}) = \bar{E}_t^{(B)}$, whenever $L_t^{(B)} > 0$ this normalization ensures $\eta_t \in (0, 2/L_t^{(B)})$ for any $\beta_t \in (0,2)$, so Lemma 3.1 applies with smoothness $L_t^{(B)}$ and guarantees per-step descent. If $L_t^{(B)} = 0$ (equivalently, $\bar{\mathbf{C}}_t^{(B)} = 0$ and $\lambda_t = 0$), then $\bar{\mathbf{N}}_t^{(B)} = 0$ as well and the update is a no-op. Crucially, because we optimize the *window average*, both $\bar{\mathbf{N}}_t^{(B)}$ and $\bar{E}_t^{(B)} = \mathrm{tr}(\bar{\mathbf{C}}_t^{(B)})$ are averages rather than sums, so their typical scale does not grow with the nominal window size B. If the write feature itself is RMS-normalized (e.g., ϕ is the identity or approximately norm-preserving), then $\bar{E}_t^{(B)} \approx d_x$ for any B; for a generic kernel map ϕ, the correct statement is simply that $\bar{E}_t^{(B)}$ tracks the realized average write-feature energy. Consequently, neither the injection $\eta_t\bar{\mathbf{N}}_t^{(B)}$ nor the decay fraction $\alpha_t := \eta_t\lambda_t$ is systematically amplified by increasing B. If the scale-coupled ridge parameterization of Section 4.1 is enabled, replace λ_t throughout this subsection by $\lambda_t^{\mathrm{eff}} := \bar{\lambda}_t\,\bar{E}_t^{(B)}$. In the current implementation, this window-average energy is used as a statistics-only multiplier when constructing λ_t^{eff} (detached / stop-gradient through the multiplier), while the step-size denominator still uses the live $\bar{E}_t^{(B)}$. Importantly, one need not materialize the $d_x \times d_x$ matrix $\mathbf{C}_t^{(B)}$ to apply the gradient: if we stack the window write-features into $\mathbf{X}_t \in \mathbb{R}^{d_x\times|\mathcal{I}_t|}$, then

$$\bar{\mathbf{C}}_t^{(B)}\mathbf{S}_{t-1} = \frac{1}{B_t}\mathbf{X}_t(\mathbf{X}_t^\top\mathbf{S}_{t-1}), \tag{4.14}$$

which can be evaluated in $O(Bd_xd_v)$ time for window size B. For the implemented positive-decay recurrence, define

$$\gamma_t := 1-\alpha_t, \qquad \alpha_t := \min(\eta_t\lambda_t,\, 1-\varepsilon_\gamma), \qquad c_0 := 1, \qquad c_t := \prod_{r=1}^{t}\gamma_r, \qquad \widetilde{\mathbf{S}}_t := \mathbf{S}_t/c_t.$$

Then the implemented positive-decay recurrence, obtained from Eq. (4.13) by replacing $1-\eta_t\lambda_t$ with γ_t, is equivalent to

$$\widetilde{\mathbf{S}}_t = \widetilde{\mathbf{S}}_{t-1} + \frac{\widehat{\eta}_t}{B_t}\sum_{j\in\mathcal{I}_t}\mathbf{x}_j\Big(\frac{\mathbf{v}_j}{c_{t-1}} - \widetilde{\mathbf{S}}_{t-1}^\top\mathbf{x}_j\Big)^\top, \qquad \widehat{\eta}_t := \eta_t/\gamma_t. \tag{4.15}$$

Within a chunk $[a,b]$, we use local prefixes $\delta_0^{(k)} := 1$ and $\delta_i^{(k)} := \prod_{r=a}^{a+i-1}\gamma_r$ for $i = 1,\dots,b-a+1$, so the incoming boundary state $\mathbf{S}_{a-1}$ is already in the local renormalized scale (empty product $= 1$). Thus, every target participating in the step-t window shares the same inverse-decay factor c_{t-1}^{-1} in global notation, or equivalently $(\delta_{i-1}^{(k)})^{-1}$ inside chunk k when $t = a+i-1$. Algorithm 4 implements this identity chunk-locally via log-prefix decays, which explains why it rescales the entire t-th window block of values by one common factor. For larger windows and for maximal compatibility with SSD-style chunk parallelism, the Inner Product formulation below is even cheaper since it avoids this term entirely.

Boundary note. Algorithm 3 is written for a fresh sequence with $\mathbf{x}_1 = \mathbf{0}$. For exact continuation on a later segment, one must also carry the final FIFO buffer $\mathcal{W}_T$; a minimal boundary payload is the last $B-1$ causal pairs. Passing only $\mathbf{S}_T$ resets the sliding objective near the boundary.

Algorithm 4 FALCON-3 via ParallelFlow (chunk-parallel forward)

Require: Queries $\{\mathbf{q}_t\}_{t=1}^T$, keys $\{\mathbf{k}_t\}_{t=0}^T$ (with BOS key $\mathbf{k}_0$), values $\{\mathbf{v}_t\}_{t=1}^T$, feature map ϕ (default identity), window size B, gains $\beta_t \in (0,2)$, ridge $\lambda_t \ge 0$, stabilizer $\varepsilon > 0$, decay clamp $\varepsilon_\gamma > 0$, chunk size C (assume $C \mid T$), init $\mathbf{S}_0$.

Ensure: Outputs $\mathbf{O} \in \mathbb{R}^{T\times d_v}$ with row t equal to $\mathbf{o}_t^\top = \phi(\mathbf{q}_t)^\top\mathbf{S}_t$ (read-after-write), final matrix state $\mathbf{S}_T$, and boundary tail $\mathcal{T}_T := \{(\mathbf{x}_j,\mathbf{v}_j)\}_{j=\max(2,T-B+2)}^T$ for exact continuation across a later segment.

Write/query features: $\mathbf{x}_t \leftarrow \phi(\mathbf{k}_{t-1})$ and $\mathbf{q}_t \leftarrow \phi(\mathbf{q}_t)$ for all t; set $\mathbf{x}_1 \leftarrow \mathbf{0}$.

Window indices $\mathcal{I}_t \leftarrow \{j \mid \max(2, t-B+1) \le j \le t\}$, window sizes $B_t \leftarrow |\mathcal{I}_t|$, and $B_t^+ \leftarrow \max(1, B_t)$.

Window-average energies: $\bar{E}_1^{(B)} \leftarrow 0$ and for $t \ge 2$ set $\bar{E}_t^{(B)} \leftarrow \frac{1}{B_t}\sum_{j\in\mathcal{I}_t}\|\mathbf{x}_j\|_2^2$.

Step sizes: set $\eta_1 \leftarrow 0$ and for $t \ge 2$ set $\eta_t \leftarrow \beta_t/(\bar{E}_t^{(B)} + \lambda_t + \varepsilon)$.

Clamp decays: $\alpha_t \leftarrow \min(\eta_t\lambda_t,\, 1-\varepsilon_\gamma)$, $\log\gamma_t \leftarrow \mathrm{log1p}(-\alpha_t)$, $\gamma_t \leftarrow \exp(\log\gamma_t)$, and $\widehat{\eta}_t \leftarrow \eta_t/\gamma_t$.

Partition into $M = T/C$ chunks $[a_k, b_k] = [(k-1)C+1, kC]$.

▷ **Phase 1: Intra-chunk ParallelFlow solves (parallel over chunks)**

for $k = 1$ **to** M **in parallel do**

  $a \leftarrow a_k$, $b \leftarrow b_k$, $p \leftarrow \max(2, a-B+1)$, $J \leftarrow \{p,\dots,b\}$.

  Slice $\mathbf{Q}^{(k)} \leftarrow (\mathbf{q}_t^\top)_{t=a}^b \in \mathbb{R}^{C\times d_x}$ and gather the write pairs $\{(\mathbf{x}_j,\mathbf{v}_j)\}_{j\in J}$.

  Build zero-padded window stacks $\mathbf{X}_{\mathrm{win}}^{(k)} \in \mathbb{R}^{d_x\times(CB)}$ and $\mathbf{V}_{\mathrm{win}}^{(k)} \in \mathbb{R}^{d_v\times(CB)}$: the i-th block of B columns contains the pairs from $\mathcal{I}_{a+i-1}$ (in increasing j), padded with $\mathbf{0}$ to width B.

  Local log-prefix decays (length-C scan): $\delta_0^{(k)} \leftarrow 1$, $u_0 \leftarrow 0$, and for $i = 1{:}C$ set $u_i \leftarrow u_{i-1} + \log\gamma_{a+i-1}$, $\delta_i^{(k)} \leftarrow \exp(u_i)$.

  Rescale values blockwise: divide every column in block i of $\mathbf{V}_{\mathrm{win}}^{(k)}$ by the scalar $\delta_{i-1}^{(k)}$, yielding $\widehat{\mathbf{V}}_{\mathrm{win}}^{(k)}$.

  ▷ Flatten time×rank. Same-time rank components are uncoupled; only earlier time-blocks interact inside `tensorInv`.

  Driver matrices: scale block i of $\mathbf{X}_{\mathrm{win}}^{(k)}$ by $s_i := \widehat{\eta}_{a+i-1}/B_{a+i-1}^+$ to obtain $\mathbf{A}^{(k)} \in \mathbb{R}^{d_x\times(CB)}$; set $\mathbf{R}^{(k)} \leftarrow -\mathbf{X}_{\mathrm{win}}^{(k)}$ and $\widetilde{\mathbf{R}}^{(k)} \leftarrow \widehat{\mathbf{V}}_{\mathrm{win}}^{(k)}$.

  $(\mathbf{W}^{(k)}, \mathbf{U}^{(k)}) \leftarrow \mathtt{tensorInv}\big(\mathbf{A}^{(k)}, \mathbf{R}^{(k)}, \widetilde{\mathbf{R}}^{(k)}\big)$ ▷ block-strict-causal solve on (time, rank) pairs

  Optional scan form: $\mathbf{M}^{(k)} \leftarrow \delta_C^{(k)}\big(\mathbf{I}_{d_x} + \mathbf{A}^{(k)}\mathbf{W}^{(k)}\big)$ and $\mathbf{b}^{(k)} \leftarrow \delta_C^{(k)}\mathbf{A}^{(k)}\mathbf{U}^{(k)}$.

  Cache $\big(\mathbf{Q}^{(k)}, \mathbf{A}^{(k)}, \mathbf{W}^{(k)}, \mathbf{U}^{(k)}, \boldsymbol{\delta}^{(k)}\big)$ where $\boldsymbol{\delta}^{(k)} := (\delta_1^{(k)},\dots,\delta_C^{(k)})$.

end for

▷ **Phase 2: Inter-chunk boundary propagation (sequential; equivalently an associative scan over $(\mathbf{M}^{(k)}, \mathbf{b}^{(k)})$)**

$\mathbf{S}_{\text{in}}^{(1)} \leftarrow \mathbf{S}_0$.

for $k = 1$ **to** M **do**

  Store $\mathbf{S}_{\text{in}}^{(k)}$.

  $\mathbf{Z}^{(k)} \leftarrow \mathbf{U}^{(k)} + \mathbf{W}^{(k)}\mathbf{S}_{\text{in}}^{(k)}$.

  $\widehat{\mathbf{S}}_{\text{out}}^{(k)} \leftarrow \mathbf{S}_{\text{in}}^{(k)} + \mathbf{A}^{(k)}\mathbf{Z}^{(k)}$. ▷ chunk exit in the local renormalized domain

  $\mathbf{S}_{\text{in}}^{(k+1)} \leftarrow \delta_C^{(k)}\, \widehat{\mathbf{S}}_{\text{out}}^{(k)}$. ▷ restore original scale at the chunk boundary; equiv. $\mathbf{S}_{\text{in}}^{(k+1)} = \mathbf{M}^{(k)}\mathbf{S}_{\text{in}}^{(k)} + \mathbf{b}^{(k)}$

end for

$\mathbf{S}_T \leftarrow \mathbf{S}_{\text{in}}^{(M+1)}$.

▷ **Phase 3: Materialize token outputs (parallel over chunks)**

for $k = 1$ **to** M **in parallel do**

  $\mathbf{Z}^{(k)} \leftarrow \mathbf{U}^{(k)} + \mathbf{W}^{(k)}\mathbf{S}_{\text{in}}^{(k)}$.

  $\mathbf{H}^{(k)} \leftarrow \mathbf{Q}^{(k)}\mathbf{S}_{\text{in}}^{(k)}$ ▷ $C \times d_v$

  $\mathbf{P}^{(k)} \leftarrow \mathbf{Q}^{(k)}\mathbf{A}^{(k)}$ ▷ $C \times (CB)$

  Define the block-causal mask $L^{(k)} \in \{0,1\}^{C\times(CB)}$ by $L^{(k)}_{i,(m-1)B+1:mB} = \mathbb{I}[m \le i]\mathbf{1}_{1\times B}$ (read-after-write).

  $\widehat{\mathbf{O}}^{(k)} \leftarrow \mathbf{H}^{(k)} + \big(\mathbf{P}^{(k)} \odot L^{(k)}\big)\mathbf{Z}^{(k)}$. ▷ local-renormalized outputs

  Row-rescale: $\mathbf{O}^{(k)}_{i,:} \leftarrow \delta_i^{(k)}\, \widehat{\mathbf{O}}^{(k)}_{i,:}$ for $i = 1{:}C$, and write into rows $a_k{:}b_k$ of $\mathbf{O}$.

end for

$\mathcal{T}_T \leftarrow \{(\mathbf{x}_j, \mathbf{v}_j)\}_{j=\max(2,T-B+2)}^{T}$.

return $(\mathbf{O}, \mathbf{S}_T, \mathcal{T}_T)$.

Chunk-boundary note. In Algorithm 4, the overlap set J in Phase 1 is the offline equivalent of carrying the last $B-1$ causal pairs across boundaries. Passing only the chunk boundary matrix state $\mathbf{S}_{\text{in}}^{(k)}$ between disjoint segments is not exact for FALCON-3, exact continuation also requires the boundary tail $\mathcal{T}$.

Associative chunk map. To make the scan claim explicit, define for chunk k

$$\mathbf{M}^{(k)} := \delta_C^{(k)}\big(\mathbf{I}_{d_x} + \mathbf{A}^{(k)}\mathbf{W}^{(k)}\big), \qquad \mathbf{b}^{(k)} := \delta_C^{(k)}\mathbf{A}^{(k)}\mathbf{U}^{(k)}.$$

Then the boundary update is the affine map

$$\mathbf{S}_{\text{in}}^{(k+1)} = \mathbf{M}^{(k)}\mathbf{S}_{\text{in}}^{(k)} + \mathbf{b}^{(k)}.$$

These chunk maps compose associatively,

$$(\mathbf{M}_2, \mathbf{b}_2) \circ (\mathbf{M}_1, \mathbf{b}_1) = (\mathbf{M}_2\mathbf{M}_1,\ \mathbf{M}_2\mathbf{b}_1 + \mathbf{b}_2),$$

so Phase 2 may be implemented either as the simple sequential loop shown in Algorithm 4 or as an associative scan over chunk maps.

4.5 Mini-batch Update Rule for Inner Product Loss (FALCON-3A)

We apply the same sliding-window principle to the Inner Product objective. To avoid coupling the update scale to the nominal window size B (a mini-batch scaling effect), we use the *window-average*

objective. For $t \geq 2$, let $B_t := |\mathcal{I}_t| \leq B$ (and skip the boundary update at $t = 1$). The windowed loss is:

$$\ell_t^{\mathrm{ip,(B)}}(\mathbf{S}_{t-1}) = -\frac{1}{B_t} \sum_{j \in \mathcal{I}_t} \langle \mathbf{S}_{t-1}^\top \mathbf{x}_j, \mathbf{v}_j \rangle + \frac{\lambda_t}{2} \|\mathbf{S}_{t-1}\|_F^2. \tag{4.16}$$

Define the window-averaged cross-covariance

$$\bar{\mathbf{N}}_t^{(B)} := \frac{1}{B_t} \sum_{j \in \mathcal{I}_t} \mathbf{x}_j \mathbf{v}_j^\top = \frac{1}{B_t} \mathbf{N}_t^{(B)}, \qquad \mathbf{N}_t^{(B)} := \sum_{j \in \mathcal{I}_t} \mathbf{x}_j \mathbf{v}_j^\top.$$

Then the gradient is $\nabla_{\mathbf{S}} \ell_t^{\mathrm{ip,(B)}} = -\bar{\mathbf{N}}_t^{(B)} + \lambda_t \mathbf{S}_{t-1}$.

Why the naive (η_t, λ_t) parameterization is poor. If one writes the update as

$$\mathbf{S}_t \;=\; (1 - \eta_t \lambda_t) \mathbf{S}_{t-1} + \eta_t \bar{\mathbf{N}}_t^{(B)}, \tag{4.17}$$

then the recurrence is governed by two quantities:

$$\rho_t := \eta_t \qquad \text{and} \qquad \alpha_t := \eta_t \lambda_t.$$

Only ρ_t (the write coefficient) and α_t (the decay fraction) matter dynamically. Parameterizing a raw ridge λ_t is therefore indirect and, worse, couples forgetting to the current write magnitude: increasing λ_t both strengthens forgetting via α_t and shrinks the current write via the induced reduction in η_t. For the inner-product objective, this coupling is unnecessary, because the feature-energy term is used only to stabilize write magnitude rather than to match a curvature term in the loss.

FALCON-3A parameterization.

$$\bar{E}_t^{(B)} := \frac{1}{B_t} \sum_{j \in \mathcal{I}_t} \|\mathbf{x}_j\|_2^2, \qquad \rho_t := \frac{\beta_t}{\bar{E}_t^{(B)} + \varepsilon_\eta}, \qquad \beta_t > 0,\ \varepsilon_\eta > 0, \tag{4.18}$$

and

$$\alpha_t := (1 - \varepsilon_\gamma)\, \sigma(\widetilde{\alpha}_t), \qquad \gamma_t := 1 - \alpha_t \in [\varepsilon_\gamma, 1), \qquad \varepsilon_\gamma > 0. \tag{4.19}$$

For learned interior steps $t \geq 2$, this parameterization yields $\alpha_t \in (0, 1 - \varepsilon_\gamma)$ and hence $\gamma_t \in (\varepsilon_\gamma, 1)$. We reserve the exact no-decay case $\alpha_t = 0$ for hard boundary / ablation settings. In particular, for the boundary no-op we set $(\rho_1, \alpha_1) = (0, 0)$, so $\gamma_1 = 1$. The FALCON-3A recurrence is then

$$\mathbf{S}_t = \gamma_t \mathbf{S}_{t-1} + \rho_t\, \bar{\mathbf{N}}_t^{(B)}. \tag{4.20}$$

This cleanly decouples plasticity from forgetting: ρ_t determines the current write amplitude, while α_t determines the multiplicative carry.

Relation to the ridge objective. The recurrence in Eq. (4.20) is exactly a gradient step on a *derived* windowed inner-product objective whose ridge is

$$\lambda_t^{\mathrm{eff}} := \alpha_t / \rho_t \qquad (\rho_t > 0).$$

Whenever a step is forced to be a strict no-op, we set $(\rho_t, \alpha_t) = (0, 0)$ (hence $\gamma_t = 1$) and do not invoke λ_t^{eff}. For $\rho_t > 0$, indeed,

$$\mathbf{S}_t = \mathbf{S}_{t-1} - \rho_t \Big(- \bar{\mathbf{N}}_t^{(B)} + \lambda_t^{\mathrm{eff}} \mathbf{S}_{t-1} \Big),$$

so Eq. (4.20) is OGD with step size ρ_t on

$$-\langle \mathbf{S}, \bar{\mathbf{N}}_t^{(B)} \rangle_F + \frac{\lambda_t^{\text{eff}}}{2} \|\mathbf{S}\|_F^2.$$

Thus, the mathematically relevant control variable is the decay fraction

$$\alpha_t = \rho_t \lambda_t^{\text{eff}},$$

not a separately parameterized raw ridge. For every interior step with $\rho_t > 0$ and $\alpha_t \in [0, 1)$, Eq. (4.20) is therefore a descent step on this derived quadratic objective: its smoothness is λ_t^{eff} and the effective step-size product is $\rho_t \lambda_t^{\text{eff}} = \alpha_t < 2$. When $\alpha_t = 0$, the objective becomes linear and the update is purely additive.

Scale properties. Because we optimize the *window average*, both $\bar{\mathbf{N}}_t^{(B)}$ and $\bar{E}_t^{(B)}$ are averages rather than sums, so their typical scale does not grow with the nominal window size B. If the write feature itself is RMS-normalized (e.g., ϕ is the identity or approximately norm-preserving), then $\bar{E}_t^{(B)} \approx d_x$ for any B; for a generic kernel map ϕ, the correct statement is simply that $\bar{E}_t^{(B)}$ tracks the realized average write-feature energy. Consequently, neither the write coefficient $\rho_t \bar{\mathbf{N}}_t^{(B)}$ nor the decay fraction α_t is systematically amplified by increasing B. Moreover, under RMS-normalized write features, ρ_t scales as $O(d_x^{-1})$, i.e. the natural width-normalized scaling for the additive write path.

We again adopt the convention $\rho_t := 0$ when the denominator in Eq. (4.18) vanishes (which can occur only in degenerate boundary cases if $\varepsilon_\eta = 0$); in that forced no-op case we also set $\alpha_t := 0$, so $\gamma_t = 1$. In practice we take $\varepsilon_\eta > 0$.

Away from boundaries (so $B_t = B$) and with no decay ($\alpha_t = 0$, equivalently $\gamma_t = 1$), each token appears in exactly B consecutive window-averages with weight $1/B$, so in the stationary case with constant $\rho_t = \rho$ its cumulative injection coefficient is $\sum_{s=j}^{j+B-1} \rho/B = \rho$, independent of B. For $B = 1$ and $t \geq 2$ (so $B_t = 1$), $\bar{\mathbf{N}}_t^{(B)} = \mathbf{x}_t \mathbf{v}_t^\top$ and Eq. (4.20) reduces to the non-sliding inner-product write with a decoupled decay. In the additive ablation $\alpha_t \equiv 0$ (or in the limit $\alpha_t \to 0$), the update is purely additive. The same normalization is still useful as a magnitude stabilizer, but it does not correspond to descent on a bounded objective.

This effectively adds the averaged evidence from the current window to the state at every step, balanced by a decay factor. The windowing ensures that each token participates in at most B *direct* write interactions, but its effect can persist beyond those B writes through the multiplicative carry $\prod_r \gamma_r$. Thus, the mechanism yields a controllable *effective* memory horizon rather than strict local support; strict locality would require explicit truncation (or a hard zero-carry override $\gamma_t = 0$, which lies outside the default parameterization $\gamma_t \geq \varepsilon_\gamma > 0$).

Streaming boundary state. Exactly as in FALCON-3, the matrix state $\mathbf{S}_t$ alone is not sufficient to continue FALCON-3A across a segment boundary. Exact continuation requires the last $B-1$ causal pairs, or an equivalent FIFO / rolling buffer from which the active-window cross-covariance and write-energy can be updated exactly. Aggregated statistics such as $(\bar{\mathbf{N}}_t^{(B)}, \bar{E}_t^{(B)}, B_t)$ alone are not sufficient for indefinite continuation, because the next step must remove the oldest contribution exactly. The masked-attention unrolling in Section 4.6 assumes this overlap is available.

Algorithm 5 Chunk-parallel masked attention for FALCON-3A (Forward)

Require: Query features $\mathbf{Q} \in \mathbb{R}^{L\times d_x}$ (row t is $\phi(\mathbf{q}_t)^\top$), write features $\mathbf{X} \in \mathbb{R}^{L\times d_x}$ (row t is $\mathbf{x}_t^\top = \phi(\mathbf{k}_{t-1})^\top$), values $\mathbf{V} \in \mathbb{R}^{L\times d_v}$ (row t is $\mathbf{v}_t^\top$), write coefficients $\boldsymbol{\rho} \in \mathbb{R}^L_{\geq 0}$, decay fractions $\boldsymbol{\alpha} \in [0, 1-\varepsilon_\gamma]^L$, window size B, chunk size C (assume $C \mid L$), decay floor $\varepsilon_\gamma > 0$, initial state $\mathbf{S}_{\text{init}} \in \mathbb{R}^{d_x\times d_v}$.

Ensure: Outputs $\mathbf{O} \in \mathbb{R}^{L\times d_v}$ where row t is $\mathbf{o}_t^\top = \phi(\mathbf{q}_t)^\top \mathbf{S}_t$ (equivalently, $\mathbf{o}_t = \mathbf{S}_t^\top \phi(\mathbf{q}_t)$) under $\mathbf{S}_t = \gamma_t \mathbf{S}_{t-1} + \rho_t \bar{\mathbf{N}}_t^{(B)}$ with $\bar{\mathbf{N}}_t^{(B)} = \frac{1}{B_t}\sum_{j\in\mathcal{I}_t} \mathbf{x}_j \mathbf{v}_j^\top$ and $\gamma_t := 1-\alpha_t \in [\varepsilon_\gamma, 1]$.

Define $\mathcal{I}_t = \{j \mid \max(2, t-B+1) \leq j \leq t\}$ for all t and set $B_t \leftarrow |\mathcal{I}_t|$.

When forming window weights, use $B_t^+ \leftarrow \max(1, B_t)$ so that $\mathbb{I}[j \in \mathcal{I}_t]/B_t^+$ is always well-defined (and equals 0 when $B_t = 0$).

Set $\rho_1 \leftarrow 0$, $\alpha_1 \leftarrow 0$, and $\gamma_1 \leftarrow 1$ (boundary no-op). For $t \geq 2$, compute $\log \gamma_t \leftarrow \mathrm{log1p}(-\alpha_t)$ and $\gamma_t \leftarrow \exp(\log \gamma_t)$.

Partition into $M = L/C$ chunks with indices $[a_k, b_k] = [(k-1)C+1, kC]$.

Initialize output buffer $\mathbf{O} \leftarrow \mathbf{0}_{L\times d_v}$.

▷ **Phase 1: Intra-chunk precomputation (parallel)**

for $k = 1$ **to** M **in parallel do**

  $a \leftarrow a_k$, $b \leftarrow b_k$, $p \leftarrow \max(2, a-B+1)$, and let $J := \{p, p+1, \ldots, b\}$ (overlap length $|J| \leq C+B-1$).

  Slice $\mathbf{Q}^{(k)} \leftarrow \mathbf{Q}_{a:b}$, $\mathbf{X}_{\text{ext}}^{(k)} \leftarrow \mathbf{X}_J$, $\mathbf{V}_{\text{ext}}^{(k)} \leftarrow \mathbf{V}_J$, $\boldsymbol{\rho}^{(k)} \leftarrow \boldsymbol{\rho}_{a:b}$, and $\log \boldsymbol{\gamma}^{(k)} \leftarrow (\log \boldsymbol{\gamma})_{a:b}$.

  Compute log-prefix decays inside the chunk: $u_0 \leftarrow 0$ and $u_i \leftarrow u_{i-1} + \log \gamma_i^{(k)}$ for $i = 1, \ldots, C$ (scan); set $\delta_i \leftarrow \exp(u_i)$ and $g^{(k)} \leftarrow \delta_C$.

  Build the causal decay matrix $D^{(k)} \in \mathbb{R}^{C\times C}$ with $D_{i,s}^{(k)} = \mathbb{I}[s \leq i] \exp(u_i - u_s)$.

  Build the B-banded window operator $A^{(k)} \in \mathbb{R}^{C\times|J|}$ (columns indexed by $j \in J$): $A_{s,j}^{(k)} = \mathbb{I}[\, j \in \mathcal{I}_{a+s-1}\,]/B_{a+s-1}^+$.

  Local mask $M^{(k)} \leftarrow D^{(k)}\, \mathrm{Diag}(\boldsymbol{\rho}^{(k)})\, A^{(k)} \in \mathbb{R}^{C\times|J|}$.

  Intra-chunk output $\mathbf{O}_{\text{intra}}^{(k)} \leftarrow \big((\mathbf{Q}^{(k)} {\mathbf{X}_{\text{ext}}^{(k)}}^\top) \odot M^{(k)}\big)\mathbf{V}_{\text{ext}}^{(k)}$.

  Chunk bias for boundary propagation: $m_{\text{end}}^{(k)} \leftarrow M_{C,:}^{(k)}$ and $\mathbf{B}^{(k)} \leftarrow {\mathbf{X}_{\text{ext}}^{(k)}}^\top \big(\mathrm{Diag}(m_{\text{end}}^{(k)})\mathbf{V}_{\text{ext}}^{(k)}\big)$.

  Cache $g^{(k)}$, $\mathbf{B}^{(k)}$, $\boldsymbol{\delta}^{(k)} := (\delta_1, \ldots, \delta_C)$, and $\mathbf{O}_{\text{intra}}^{(k)}$.

end for

▷ **Phase 2: Inter-chunk recurrence (sequential over chunks or scan)**

$\mathbf{S}_{\text{in}}^{(1)} \leftarrow \mathbf{S}_{\text{init}}$

for $k = 1$ **to** M **do**

  $\mathbf{S}_{\text{out}}^{(k)} \leftarrow g^{(k)} \mathbf{S}_{\text{in}}^{(k)} + \mathbf{B}^{(k)}$

  $\mathbf{S}_{\text{in}}^{(k+1)} \leftarrow \mathbf{S}_{\text{out}}^{(k)}$

end for

▷ **Phase 3: Materialize outputs (parallel)**

for $k = 1$ **to** M **in parallel do**

  $H^{(k)} \leftarrow \mathbf{Q}^{(k)} \mathbf{S}_{\text{in}}^{(k)}$ ▷ $C \times d_v$

  $\mathbf{O}_{\text{hist}}^{(k)} \leftarrow \mathrm{Diag}(\boldsymbol{\delta}^{(k)})\, H^{(k)}$

  $\mathbf{O}^{(k)} \leftarrow \mathbf{O}_{\text{hist}}^{(k)} + \mathbf{O}_{\text{intra}}^{(k)}$

  Write $\mathbf{O}_{a_k:b_k} \leftarrow \mathbf{O}^{(k)}$.

end for

return $\mathbf{O}$ (concatenate $\{\mathbf{O}^{(k)}\}_{k=1}^M$).

4.6 Parallel (Attention) Form of Mini-batch Inner Product Update

We show that the recurrent sliding-window update in Section 4.5 is algebraically equivalent to an (unnormalized) dot-product attention matrix with a structured causal mask. This attention-matrix

view is useful for GPU-parallel training and is the standard unrolling trick used to connect gated linear recurrences to masked attention formulations (e.g., RetNet (Sun et al., 2023), Lightning Attention (Qin et al., 2024), Gated Linear Attention (Yang et al., 2023), and Mamba-2 (Gu and Dao, 2023; Dao and Gu, 2024)).

For the unclamped exposition, let $\gamma_s := 1 - \alpha_s$ with $\alpha_s \in [0, 1)$, so $\gamma_s \in (0, 1]$. Unrolling the recurrence

$$\mathbf{S}_s = \gamma_s \mathbf{S}_{s-1} + \rho_s \bar{\mathbf{N}}_s^{(B)}$$

to time t under read-after-write yields:

$$\mathbf{S}_t = \left(\prod_{r=1}^{t} \gamma_r \right) \mathbf{S}_0 + \sum_{s=1}^{t} \left(\prod_{r=s+1}^{t} \gamma_r \right) \rho_s \bar{\mathbf{N}}_s^{(B)}. \tag{4.21}$$

Throughout the paper we use the boundary convention $\mathbf{x}_1 = \phi(\mathbf{k}_0) = \mathbf{0}$ and set $(\rho_1, \alpha_1) = (0, 0)$ (hence $\gamma_1 = 1$), and for prefill we take $\mathbf{S}_0 = \mathbf{0}$; in that default case the sum effectively starts at $s = 2$.

Under next-latent alignment, define the write feature $\mathbf{x}_j := \phi(\mathbf{k}_{j-1})$ (so $\mathbf{x}_j = \mathbf{k}_{j-1}$ when ϕ is the identity). We follow the previous boundary convention that fast-memory updates start at $t = 2$ (equivalently, $\mathbf{x}_1 = \mathbf{0}$ so the $t = 1$ update is a no-op), so the sliding-window indices satisfy $\mathcal{I}_s \subseteq \{2, \dots, s\}$ and $M_{t,1} = 0$. Let $B_s := |\mathcal{I}_s| \le B$ and recall

$$\bar{\mathbf{N}}_s^{(B)} = \frac{1}{B_s} \sum_{j \in \mathcal{I}_s} \mathbf{x}_j \mathbf{v}_j^\top, \qquad \mathcal{I}_s = \{ j \mid \max(2, s - B + 1) \le j \le s \}.$$

Then:

$$\mathbf{S}_t = \sum_{s=2}^{t} \sum_{j=2}^{t} \mathbb{I}[j \in \mathcal{I}_s] \left(\frac{\rho_s}{B_s} \prod_{r=s+1}^{t} \gamma_r \right) \mathbf{x}_j \mathbf{v}_j^\top. \tag{4.22}$$

We swap the summation order to sum over tokens j first. For $j \ge 2$, the condition $j \in \mathcal{I}_s$ is equivalent to $s - B + 1 \le j \le s$, i.e. $j \le s \le j + B - 1$. Combining this with the outer bound $s \le t$, the validity range for the update step s given a token j is $j \le s \le \min(t, j + B - 1)$. Thus:

$$\mathbf{S}_t = \sum_{j=2}^{t} \left[\sum_{s=j}^{\min(t, j+B-1)} \frac{\rho_s}{B_s} \left(\prod_{r=s+1}^{t} \gamma_r \right) \right] \mathbf{x}_j \mathbf{v}_j^\top. \tag{4.23}$$

Defining the scalar mask $M_{t,j}$ as the term in brackets (and $M_{t,j} = 0$ for $j > t$ and for $j < 2$ under our boundary convention), the causal readout at position t is

$$\mathbf{o}_t = \mathbf{S}_t^\top \phi(\mathbf{q}_t) = \sum_{j=2}^{t} M_{t,j} \langle \phi(\mathbf{q}_t), \mathbf{x}_j \rangle \mathbf{v}_j. \tag{4.24}$$

Equivalently, stacking query features $\mathbf{Q} \in \mathbb{R}^{L \times d_x}$ with row t equal to $\phi(\mathbf{q}_t)$ and write features $\mathbf{X} \in \mathbb{R}^{L \times d_x}$ (row j is $\mathbf{x}_j$), and values into $\mathbf{V} \in \mathbb{R}^{L \times d_v}$, and defining $\mathbf{M} \in \mathbb{R}^{L \times L}$ with entries $\mathbf{M}_{t,j} = M_{t,j}$ for $2 \le j \le t$ (and 0 otherwise), the sequence-level readout is $\mathbf{O} = (\mathbf{Q}\mathbf{X}^\top \odot \mathbf{M})\mathbf{V}$. This is a masked attention form with a *gate-dependent* (input-conditioned) causal mask induced by the sliding-window mini-batch and multiplicative decay: each coefficient $M_{t,j}$ is a sum of at most B

decayed write coefficients ρ_s/B_s, and it is generally nonzero for all $j \le t$. Importantly, $M_{t,j}$ need not be monotone in the lag $t-j$: because token j is injected into $\mathbf{N}_s^{(B)}$ for $s = j, \dots, j+B-1$, weights within the most recent B positions can increase as j moves backward (older tokens are re-added more times), while tokens farther in the past decay geometrically when $\gamma_r \in (0,1)$. In the stationary case $\rho_s = \rho$ and $\gamma_s = \gamma \in (0,1)$,

$$M_{t,j} = \frac{\rho}{B} \sum_{u=\max(0,\, t-j-B+1)}^{t-j} \gamma^u,$$

which makes the moving-sum then exponential-tail structure explicit. All expressions above follow our paper-wide read-after-write convention: the readout at position t uses the updated state $\mathbf{S}_t$ and is used to predict token $t{+}1$.

Mask Structure. The mask $M_{t,j}$ represents the accumulated effective learning rate for token pair j by time t:

$$M_{t,j} = \sum_{s=j}^{\min(t,j+B-1)} \frac{\rho_s}{B_s} \prod_{r=s+1}^{t} (1-\alpha_r), \tag{4.25}$$

This mask cleanly separates the window-average accumulation from the global multiplicative carry $\gamma_t = 1-\alpha_t$ and is convenient for GPU implementation.

Log-space cumulative decay. Directly forming $\prod_{r=s+1}^{t} \gamma_r$ can underflow over long sequences, especially in reduced precision. Define the log cumulative decay

$$\zeta_0 := 0, \qquad \zeta_t := \zeta_{t-1} + \log \gamma_t \quad (t \ge 1),$$

so that $\prod_{r=s+1}^{t} \gamma_r = \exp(\zeta_t - \zeta_s)$ and

$$M_{t,j} = \sum_{s=j}^{\min(t,j+B-1)} \frac{\rho_s}{B_s} \exp(\zeta_t - \zeta_s). \tag{4.26}$$

For improved accuracy when α_t is small, compute $\log \gamma_t = \log(1-\alpha_t)$ as $\mathrm{log1p}(-\alpha_t)$ (in fp32).

Chunk-local renormalization. Even with the log representation, explicitly materializing global ζ_t and then forming $\exp(\zeta_t - \zeta_s)$ at very large lags can underflow in reduced precision (and is unnecessary for chunked kernels). All chunk-parallel algorithms in this paper, therefore, reset the log-prefix to 0 at chunk boundaries, use only *within-chunk* differences $u_i - u_s$ with $u_0 = 0$, and propagate boundary states via the chunk-exit decay $g^{(k)} = \prod_{t \in \text{chunk } k} \gamma_t$. This is exactly the local log-decay trick used by Algorithm 5 (and its backward pass in Appendix B.1).

Chunk-parallel evaluation. For implementation, it is helpful to expose the mask as the composition of (i) a causal decay kernel and (ii) a B-banded moving-average window operator. Define

$$D_{t,s} := \mathbb{I}[s \le t] \prod_{r=s+1}^{t} \gamma_r, \qquad B_s^+ := \max(1, B_s), \qquad A_{s,j} := \frac{\mathbb{I}[j \in \mathcal{I}_s]}{B_s^+}, \qquad \mathbf{M} = \mathbf{D}\ \mathrm{Diag}(\boldsymbol{\rho})\, \mathbf{A},$$

so $\mathbf{M}_{t,j} = \sum_s D_{t,s} \rho_s A_{s,j}$ recovers Eq. (4.26). Under our boundary convention ($\mathcal{I}_1 = \emptyset$ and $\rho_1 = 0$), the $s = 1$ row of $\mathbf{A}$ is all zeros, and for all $s \ge 2$ we have $B_s^+ = B_s$, so this agrees with $A_{s,j} = \mathbb{I}[j \in \mathcal{I}_s]/B_s$

while avoiding a 0/0 definition at $s = 1$. This yields a three-phase chunked algorithm: per-chunk precomputation (parallel), inter-chunk boundary propagation (short recurrence/scan), and final output materialization (parallel).

Backward pass. A numerically stable backward pass for Algorithm 5, differentiating through both the structured mask and the chunk-local log-decay renormalization, is given in Appendix B.1 (Algorithm 6).

Relation to test-time training and Titans-style memory objectives. Test-Time Training (TTT) updates a subset of model parameters at inference by taking gradient steps on an internal, self-supervised objective, and recent work formalizes this mechanism as *test-time regression* in a feature space (Sun et al., 2024; Wang et al., 2025). Our fast-weight update is a constrained, single-step instance of this paradigm: the recurrent state $\mathbf{S}$ is the adapted parameter and Eq. (3.1) is the inner objective. Relative to generic TTT, we enforce strict autoregressive causality by updating only after $\mathbf{v}_t$ is revealed and by writing it under the prefix feature available at prediction time, $\mathbf{x}_t = \phi(\mathbf{k}_{t-1})$. Titans and ATLAS similarly view the recurrent state as fast memory optimized online with an internal objective over the stream (Behrouz et al., 2024, 2025a), our contribution is to make the key-value alignment explicit and to derive normalized first-order rules and sliding-window variants compatible with SSD-style chunk-parallel training.

5 Experiments

Our main empirical evaluation is language modeling at approximately 124M-130M parameters. We train all models on FineWeb-Edu with a matched 50B-token budget and evaluate both held-out perplexity and downstream task accuracy. We also use variable-length multi-digit addition as a supporting diagnostic, since it isolates causal storage and length extrapolation in a controlled setting. Unless a table row states otherwise, fast-weight models use the scaled formulation of Section 4.1, and next-latent alignment is implemented by shifting the write-key stream by one position with a zero BOS write feature, so the first write is a no-op. For FALCON-2A, row labels *ctx*β and *ctx*η are run identifiers indicating whether the learned control is interpreted as the NLMS gain β_t or as the post-normalization write coefficient η_t; the theory in Section 4.3 is written in the gain-parameterized form Eq. (4.8). The descent discussion there applies directly to the gain-parameterized form; direct-η_t runs should be read as empirical ablations unless one separately enforces $\eta_t \lambda_t < 2$ for all steps with $\lambda_t > 0$. For FALCON-3A, the actual recurrence always uses the decoupled (ρ_t, α_t) parameterization of Section 4.5; labels such as *ctx*η-*ctx*λ are legacy run names and should be read only as context-dependent plasticity/decay tags, not as literal (η_t, λ_t) state variables. QK-RMSNorm and QK-ℓ_2 denote the query/key normalization used inside the fast-weight block.

5.1 Language modeling experiments

We train 124M-130M parameter models for 100,000 optimization steps with sequence length 1,024 and global batch size 480, for a total budget of approximately 49.2B tokens (about 50B). The Transformer baseline uses a LLaMA-style architecture with RoPE and SwiGLU. Recurrent baselines include RetNet/LightningAttn, Mamba-2, DeltaNet, and Gated DeltaNet. For perplexity, the completed FALCON runs at this scale are the FALCON-2A variants; for downstream transfer, Table 2 additionally includes the completed FALCON-3A.3 checkpoint. Unless explicitly ablated, fast-weight models use QK-RMSNorm and lightweight short convolutions on the attention projections.

For these runs, all models are trained in bfloat16 with AdamW, tied input/output embeddings, Pre-Norm RMSNorm, no bias terms, and no dropout. We use μP-style width scaling, base learning rate 10^{-3} with cosine decay and 2,000 warmup steps, $(\beta_1, \beta_2) = (0.9, 0.95)$, weight decay 0.1, and gradient clipping at 1.0. All runs are performed on a single 4-GPU node (H100 or H200). We report teacher-forced perplexity and zero-shot / one-shot downstream accuracy.

On validation set perplexity, shown in Table 1, FALCON is competitive but not uniformly stronger than the best baselines. Among our variants, FALCON-2A.3 (QK-RMSNorm, ctxη-ctxλ) gives the best perplexity, reaching 17.64 on FineWeb-Edu validation, while Gated DeltaNet remains strongest overall among the models compared here. Table 2 shows that the same next-latent update also transfers reasonably well to downstream evaluation: FALCON-2A.2 achieves the best zero-shot average in this comparison, and its one-shot average remains competitive with Mamba-2 and the Transformer. Within the FALCON family, QK-RMSNorm improves perplexity relative to QK-ℓ_2 normalization, while ctxη improves average downstream accuracy over the corresponding ctxβ variant. The main takeaway is therefore not a uniform win at this scale, but that our autoregressively aligned, normalized fast-weight updates preserve language-model quality under a realistic pretraining budget.

Table 1 Comparison of models with 124M-130M parameters trained for a 50B-token budget on FineWeb-Edu. Lower is better.

Model	Perplexity ↓		
	Wiki.	LMB.	FineEdu.
124M-130M parameters			
(124M) Transformer (w. RoPE)	33.25	47.43	17.38
(130M) RetNet/LightningAttn	36.86	65.16	18.79
(130M) Mamba-2	34.53	48.74	17.70
(130M) DeltaNet	34.19	52.84	17.84
(130M) Gated DeltaNet	30.99	46.70	17.32
Ours			
(130M) **FALCON-2A.1 (QK-ℓ_2-norm, ctxβ-ctxλ)**	34.41	47.93	17.70
(130M) **FALCON-2A.2 (QK-ℓ_2-norm, ctxη-ctxλ)**	34.20	51.01	17.70
(130M) **FALCON-2A.3 (QK-RMSNorm, ctxη-ctxλ)**	34.02	49.84	17.40
(130M) **FALCON-3A.3 (QK-RMSNorm, ctxη-ctxλ)**	34.08	53.38	17.59

5.2 Variable-length arithmetic addition tasks

We use variable-length multi-digit addition as a controlled stress test for causal storage and extrapolation, following Kaiser and Sutskever (2015). Each sample presents an n-digit prompt

$$s_{\text{in}} := \texttt{"}\text{digits}_n(a) \texttt{ + } \text{digits}_n(b) \texttt{ ="},$$

and asks the model to generate the reversed $(n{+}1)$-digit sum

$$s_{\text{out}} := \texttt{" }\text{rev}\big(\text{digits}_{n+1}(a+b)\big)\texttt{"},$$

Table 2 Language-model transfer to downstream tasks. Zero-shot and one-shot accuracy. Accuracies use `acc`; tasks marked with * use `acc_n` as in `lm-evaluation-harness`. Avg. is the unweighted average across the 8 tasks.

Model	Accuracy ↑								
	PIQA	Hella.*	Wino.	ARC-e	ARC-c*	OBQA*	Social IQA	SciQ	Avg.
Zero-shot (0-shot)									
124M-130M parameters									
(124M) Transformer (w. RoPE)	65.67	37.54	51.70	52.36	27.65	31.60	38.84	79.90	48.16
(130M) RetNet/LightningAttn	64.91	35.36	49.64	57.62	26.28	32.20	37.97	80.40	48.05
(130M) Mamba-2	66.32	36.89	50.75	58.16	26.62	32.60	38.38	80.70	48.80
(130M) DeltaNet	66.38	37.15	52.33	57.37	26.79	34.00	39.00	78.00	48.88
(130M) Gated DeltaNet	65.51	37.75	49.72	58.88	27.90	31.60	38.28	80.60	48.78
Ours									
(130M) **FALCON-2A.1 (QK-ℓ_2-norm, ctxβ-ctxλ)**	66.05	37.27	50.67	57.62	27.65	31.60	38.95	81.10	48.86
(130M) **FALCON-2A.2 (QK-ℓ_2-norm, ctxη-ctxλ)**	67.03	37.29	52.33	57.37	25.94	33.20	38.84	82.40	49.30
(130M) **FALCON-2A.3 (QK-RMSNorm, ctxη-ctxλ)**	66.10	37.55	50.12	59.01	26.79	32.00	37.82	82.20	48.95
(130M) **FALCON-3A.3 (QK-RMSNorm, ctxη-ctxλ)**	65.34	37.30	50.99	57.37	26.37	33.60	39.41	81.60	49.00
One-shot (1-shot)									
(124M) Transformer (w. RoPE)	66.43	37.55	50.28	59.64	29.01	30.00	39.82	84.60	49.67
(130M) RetNet/LightningAttn	65.13	35.19	49.64	56.78	25.85	28.80	36.44	81.50	47.42
(130M) Mamba-2	66.70	36.61	51.07	58.63	26.96	32.40	37.97	82.90	49.16
(130M) DeltaNet	66.27	36.67	50.36	57.70	27.39	32.00	37.72	79.90	48.50
(130M) Gated DeltaNet	65.40	37.81	51.30	58.29	26.88	30.00	37.26	81.60	48.57
Ours									
(130M) **FALCON-2A.1 (QK-ℓ_2-norm, ctxβ-ctxλ)**	66.81	37.41	50.75	57.53	27.99	32.40	37.92	81.80	49.08
(130M) **FALCON-2A.2 (QK-ℓ_2-norm, ctxη-ctxλ)**	66.38	36.97	52.96	57.32	27.82	31.40	37.56	83.20	49.20
(130M) **FALCON-2A.3 (QK-RMSNorm, ctxη-ctxλ)**	65.78	37.22	49.72	58.88	27.73	31.40	37.97	82.40	48.89
(130M) **FALCON-3A.3 (QK-RMSNorm, ctxη-ctxλ)**	65.72	36.47	51.78	58.29	26.54	32.00	38.23	83.20	49.03

so that the least significant digit is produced first. We train on widths sampled uniformly from $[1, 32]$ and optimize masked next-token log-likelihood on the target suffix only. Evaluation uses teacher forcing and sequence-level exact-match accuracy; under exact match, this is equivalent to greedy decoding success on the target suffix. For length extrapolation, we hold the model fixed and sweep evaluation widths $N \in \{33, \ldots, 48\}$.

Table 3 reports both in-distribution validation accuracy and out-of-distribution exact-match accuracy averaged over 33-48 digits. FALCON-3A.3 achieves the best extrapolation performance, with 87.2 mean accuracy, followed by FALCON-2A.3 at 85.9, both outperform linear attention and the Transformer on this task. We view this experiment as supporting evidence rather than the paper's primary result: it isolates the memory-writing behavior of the recurrent state and shows that the shifted fast-memory update substantially improves length generalization when storage and carry propagation dominate.

6 Related Work

Efficient Sequence Modeling and State-Space Duality. Standard Transformers (Vaswani et al., 2017) incur quadratic $\mathcal{O}(N^2)$ compute and memory costs due to the materialization of the attention

Table 3 Teacher-forced length generalization on variable-digit addition. Higher is better.

Model	Best step	Val. acc.	Mean acc.	Acc@d33/d48
Transformer (w. RoPE)	2000	100.0	65.8	97.0/49.0
RetNet/LightningAttn	2000	99.7	82.9	99.0/63.0
Linear Attn (ctxλ)	2000	100.0	75.2	100.0/51.0
Ours				
FALCON-2A.1 (QK-ℓ_2-norm, ctxβ-ctxλ)	1900	100.0	80.6	100.0/59.0
FALCON-2A.2 (QK-ℓ_2-norm, ctxη-ctxλ)	2000	100.0	85.2	100.0/63.0
FALCON-2A.3 (QK-RMSNorm, ctxη-ctxλ)	1900	99.8	85.9	100.0/69.0
FALCON-3A.3 (QK-RMSNorm, ctxη-ctxλ)	2000	99.9	**87.2**	100.0/69.0

matrix. To address this, Linear Attention (Katharopoulos et al., 2020) reorders the matrix multiplication via kernel feature maps, effectively treating the context as a recurrent accumulation of outer products. Other subquadratic attention replacements include random-feature approximations to softmax attention, such as Performer (Choromanski et al., 2020), long-convolution operators such as Hyena (Poli et al., 2023), and retention-style recurrent attention (Sun et al., 2023) or RNN–Transformer hybrids (Peng et al., 2023). Parallel development in Structured State Space Models (SSMs), such as S4 (Gu et al., 2021) and Mamba (Gu and Dao, 2023), utilized discretized continuous-time dynamics to achieve similar linear scaling. Recently, Mamba-2 (Dao and Gu, 2024) unified these paradigms under Structured State Space Duality (SSD), proving that selective SSMs are algorithmically dual to linear attention with semi-separable masks. While Mamba-2 and related SSD-style models focus on hardware utilization (via chunk-parallel scans) and expressive discretizations, they largely retain additive or gated accumulation as the state write rule. In contrast, our work re-examines the objective that induces the state update. We derive autoregressively aligned, NLMS-stabilized regression updates and sliding-window variants that remain compatible with SSD-style chunk-parallel training.

Fast Weights and Delta Networks. The concept of Fast Weights (Schmidhuber, 1992) states that a neural network can maintain a high-capacity short-term memory matrix updated by the immediate input stream. Schlag et al. (2021) formalized this as Linear Transformers are Secretly Fast Weight Programmers, proposing the Delta Network, which updates the state via a gradient step on the prediction error rather than Hebbian addition. Yang et al. (2024a) extended this with Gated Delta Networks, introducing Mamba-style gating to improve stability. However, these prior approaches often treat the update rule as an architectural choice rather than an optimization problem.

Adaptive Filtering and Online Regression. The delta-rule updates used in fast-weight models are closely related to classical adaptive filtering, where the LMS delta rule and its normalized variant (NLMS) are standard algorithms for scale-robust online regression, and second-order methods such as recursive least squares (RLS) provide exact online ridge updates at higher cost (Sayed, 2011). Our NLMS-based FALCON-2 update can be viewed as importing these stability/normalization principles into fast-weight attention under strict causality, while remaining compatible with modern parallel implementations of linear recurrences (Yang et al., 2024b; Dao and Gu, 2024; Cirone and Salvi, 2025).

In-Context Learning as Implicit Optimization. Recent theoretical work interprets the forward pass of Transformers as performing implicit (preconditioned) gradient descent on in-context examples (Von Oswald et al., 2023; Ahn et al., 2023; Cheng et al., 2023). von Oswald et al. (2025) explicitly operationalizes this by proposing Mesa-Layers that solve a least-squares problem within the forward pass. MesaNet performs test-time optimization by solving (or approximately solving, depending on the stopping criterion) sequential linear systems via conjugate gradient, with a design explicitly aimed at chunkwise parallelism, at the cost of increased inference compute. By analogy to networking's layering-as-decomposition, our approach aligns with this layers-as-optimizers philosophy (Chiang et al., 2007) but constrains the solution to strict online updates. We formalize the state-space model as an online ridge regressor predicting the next embedding. This perspective allows us to unify the sliding-window attention mechanism with recurrent state updates, resulting in the Sliding State mechanism: a mathematically grounded method for finite-window forgetting via mini-batch SGD (Behrouz et al., 2025a), bridging the gap between local attention and global recurrence.

Context Engineering and Memory Management. Complementary to designing better attention and recurrent fast memories that are hardware efficient (Guo et al., 2025; Zhang et al., 2025b), a line of work studies *context engineering*: managing and transforming the context presented to the model. This includes KV-cache compression, quantization, or eviction schemes that reduce long-context overhead while preserving salient information (Liu et al., 2024c; Xiao et al., 2024; Zhang et al., 2025a). Other approaches view in-context learning through multi-timescale memory systems that combine short-lived activations with synaptic/plastic fast weights (Schlag et al., 2021; Behrouz et al., 2025b).

7 Conclusion

In this paper, we recast recurrent sequence modeling as online continual learning with a fast state that solves a next-latent prediction problem. Under the read-after-write convention used throughout, common fast-weight recurrences become temporally shifted by one step; we make that shift explicit by using a framework in which the state maps $\mathbf{k}_{t-1}$ to $\mathbf{v}_t$. Building on this lens, we introduce FALCON-2 (NLMS-normalized delta updates), FALCON-3 (sliding-window regression via a single mini-batch gradient step), and inner-product counterparts FALCON-2A/FALCON-3A. In the completed experiments in this draft, primarily on the inner-product variants at $\sim$130M scale, these updates are competitive with strong linear-recurrent baselines while using an explicit autoregressively aligned next-latent write convention. More broadly, our framework casts SSM-style sequence models as continual-learning systems with an explicit fast-memory objective, providing a principled handle on the stability-plasticity trade-off via learned plasticity, forgetting, and bounded rehearsal.

Acknowledgement

We thank Andy Yao, Tri Dao, Sanjeev Arora, Karthik Narasimhan, Eric Song, and Gon Buzaglo for helpful discussions and constructive feedback. We thank Princeton AI Lab and Princeton Language Intelligence for providing computing resources. Steve Ta deeply appreciates Vikram Ramaswamy for believing in him, and Vikram's support helped Steve discover his own path in machine learning. We used Large Language Models to help refine this paper. Their role was limited to improving the clarity and readability of the text.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Kwangjun Ahn, Xiang Cheng, Hadi Daneshmand, and Suvrit Sra. Transformers learn to implement preconditioned gradient descent for in-context learning. *Advances in Neural Information Processing Systems*, 36:45614–45650, 2023.

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. *Advances in neural information processing systems*, 29, 2016.

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

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

Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. Nested learning: The illusion of deep learning architectures. *arXiv preprint arXiv:2512.24695*, 2025b.

Christian Bischof and Charles Van Loan. The wy representation for products of householder matrices. *SIAM Journal on Scientific and Statistical Computing*, 8(1):s2–s13, 1987.

Xiang Cheng, Yuxin Chen, and Suvrit Sra. Transformers implement functional gradient descent to learn non-linear functions in context. *arXiv preprint arXiv:2312.06528*, 2023.

Mung Chiang, Steven H Low, A Robert Calderbank, and John C Doyle. Layering as optimization decomposition: A mathematical theory of network architectures. *Proceedings of the IEEE*, 95(1): 255–312, 2007.

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

Nicola Muca Cirone and Cristopher Salvi. Parallelflow: Parallelizing linear transformers via flow discretization. *arXiv preprint arXiv:2504.00492*, 2025.

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

Daniel Y Fu, Tri Dao, Khaled K Saab, Armin W Thomas, Atri Rudra, and Christopher Ré. Hungry hungry hippos: Towards language modeling with state space models. *arXiv preprint arXiv:2212.14052*, 2022.

Riccardo Grazzi, Julien Siems, Arber Zela, Jörg KH Franke, Frank Hutter, and Massimiliano Pontil. Unlocking state-tracking in linear rnns through negative eigenvalues. *arXiv preprint arXiv:2411.12537*, 2024.

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

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

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

Geoffrey E Hinton and David C Plaut. Using fast weights to deblur old memories. In *Proceedings of the ninth annual conference of the Cognitive Science Society*, pages 177–186, 1987.

Łukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

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

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

Zicheng Liu, Siyuan Li, Li Wang, Zedong Wang, Yunfan Liu, and Stan Z Li. Short-long convolutions help hardware-efficient linear attention to focus on long sequences. *arXiv preprint arXiv:2406.08128*, 2024b.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*, 2024c.

Bo Peng, Eric Alcaide, Quentin Anthony, Alon Albalak, Samuel Arcadinho, Stella Biderman, Huanqi Cao, Xin Cheng, Michael Chung, Matteo Grella, et al. Rwkv: Reinventing rnns for the transformer era. *arXiv preprint arXiv:2305.13048*, 2023.

Michael Poli, Stefano Massaroli, Eric Nguyen, Daniel Y Fu, Tri Dao, Stephen Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. Hyena hierarchy: Towards larger convolutional language models. In *International Conference on Machine Learning*, pages 28043–28078. PMLR, 2023.

Zhen Qin, Weigao Sun, Dong Li, Xuyang Shen, Weixuan Sun, and Yiran Zhong. Lightning attention-2: A free lunch for handling unlimited sequence lengths in large language models. *arXiv preprint arXiv:2401.04658*, 2024.

Ali H Sayed. *Adaptive filters*. John Wiley & Sons, 2011.

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

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

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

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

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, et al. Kimi linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

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

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

Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pages 35151–35174. PMLR, 2023.

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

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

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *arXiv preprint arXiv:2410.10819*, 2024.

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

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

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *arXiv preprint arXiv:2406.06484*, 2024b.

Yifan Zhang, Yifeng Liu, Huizhuo Yuan, Zhen Qin, Yang Yuan, Quanquan Gu, and Andrew Chi-Chih Yao. Tensor product attention is all you need. *arXiv preprint arXiv:2501.06425*, 2025a.

Yifan Zhang, Zhen Qin, and Quanquan Gu. Higher-order linear attention. *arXiv preprint arXiv:2510.27258*, 2025b.

Appendix

A Exact Online Ridge via Recursive Least Squares

Appendix notation. We reuse the main-text regression notation: $\mathbf{x}_t$ denotes the write feature (typically $\phi(\mathbf{k}_{t-1})$ under next-latent alignment), $\mathbf{y}_t$ the target (typically $\mathbf{v}_t$), and $\mathbf{S}_t$ the fast-weight state. The residual $\mathbf{r}_t = \mathbf{y}_t - \mathbf{S}_{t-1}^\top \mathbf{x}_t$ matches $\mathbf{r}_t$ in the main text.

Recursive least squares (RLS). RLS realizes the exact cumulative ridge-regression solution online using the matrix inversion lemma. Here we treat the constant-ridge case $\lambda > 0$, which is not the exact closed-form solution for the time-varying λ_t recurrences used elsewhere in the paper. Let

$$\mathbf{P}_{t-1} := \big(\lambda \mathbf{I} + \textstyle\sum_{s=1}^{t-1} \mathbf{x}_s \mathbf{x}_s^\top\big)^{-1}, \qquad \mathbf{P}_0 := \lambda^{-1}\mathbf{I} \;\; (\lambda > 0).$$

Define the gain and covariance updates

$$\mathbf{g}_t = \frac{\mathbf{P}_{t-1}\mathbf{x}_t}{1 + \mathbf{x}_t^\top \mathbf{P}_{t-1}\mathbf{x}_t}, \qquad \mathbf{P}_t = \mathbf{P}_{t-1} - \mathbf{g}_t \mathbf{x}_t^\top \mathbf{P}_{t-1}. \tag{A.1}$$

Then the parameter update is

$$\mathbf{r}_t := \mathbf{y}_t - \mathbf{S}_{t-1}^\top \mathbf{x}_t, \qquad \mathbf{S}_t = \mathbf{S}_{t-1} + \mathbf{g}_t\, \mathbf{r}_t^\top. \tag{A.2}$$

These recursions cost $O(d_x^2 + d_x d_v)$ per step, where $d_x := \dim(\mathbf{x}_t)$, and satisfy that $\mathbf{S}_t$ is the exact ridge-regression solution after processing t samples, i.e. the minimizer of the cumulative ridge objective on $\{(\mathbf{x}_s, \mathbf{y}_s)\}_{s=1}^t$ under the stated prior. Exact online solvers of this form are much harder to parallelize than first-order recurrences, which is why MesaNet (von Oswald et al., 2025) uses an approximate chunk-parallel approach.

B More on Mini-batch Update Rule for Inner Product Loss (FALCON-3A)

B.1 Backward pass for chunk-parallel FALCON-3A

Algorithm 6 Chunk-parallel masked attention for FALCON-3A (Backward)

Require: Upstream gradient $\bar{\mathbf{O}} \in \mathbb{R}^{L \times d_v}$ and cached forward tensors from Algorithm 5. In particular, for each chunk $k \in \{1, \dots, M\}$ (with indices $[a_k, b_k]$ and overlap set J_k), cache: $\mathbf{Q}^{(k)} = \mathbf{Q}_{a_k:b_k}$, $\mathbf{X}_{\text{ext}}^{(k)} = \mathbf{X}_{J_k}$, $\mathbf{V}_{\text{ext}}^{(k)} = \mathbf{V}_{J_k}$, $\boldsymbol{\rho}^{(k)} = \boldsymbol{\rho}_{a_k:b_k}$, $\boldsymbol{\alpha}^{(k)} = \boldsymbol{\alpha}_{a_k:b_k}$, $\log \boldsymbol{\gamma}^{(k)} = (\log \boldsymbol{\gamma})_{a_k:b_k}$, local prefixes $u_{0:C}^{(k)}$ and $\boldsymbol{\delta}^{(k)} = (\delta_1, \dots, \delta_C)$, decay kernel $\mathbf{D}^{(k)}$, window operator $\mathbf{A}^{(k)}$, mask $\mathbf{M}^{(k)}$, chunk-exit decay $g^{(k)} = \delta_C$, last-row mask $m_{\text{end}}^{(k)} = \mathbf{M}_{C,:}^{(k)}$, and boundary state $\mathbf{S}_{\text{in}}^{(k)}$.

Ensure: Gradients $\bar{\mathbf{Q}} \in \mathbb{R}^{L \times d_x}$, $\bar{\mathbf{X}} \in \mathbb{R}^{L \times d_x}$, $\bar{\mathbf{V}} \in \mathbb{R}^{L \times d_v}$, $\bar{\boldsymbol{\rho}} \in \mathbb{R}^L$, $\bar{\boldsymbol{\alpha}} \in \mathbb{R}^L$, and $\bar{\mathbf{S}}_{\text{init}} \in \mathbb{R}^{d_x \times d_v}$.

Reshape $\bar{\mathbf{O}}$ into chunks $\bar{\mathbf{O}}^{(k)} = \bar{\mathbf{O}}_{a_k:b_k}$.

Initialize all output gradients to zero. Initialize per-chunk buffers: $\bar{\mathbf{S}}_{\text{in, out}}^{(k)} \leftarrow \mathbf{0}$, $\bar{\boldsymbol{\delta}}^{(k)} \leftarrow \mathbf{0}$, $\bar{\mathbf{M}}^{(k)} \leftarrow \mathbf{0}$, $\bar{\mathbf{Q}}^{(k)} \leftarrow \mathbf{0}$, $\bar{\mathbf{X}}_{\text{ext}}^{(k)} \leftarrow \mathbf{0}$, $\bar{\mathbf{V}}_{\text{ext}}^{(k)} \leftarrow \mathbf{0}$.

▷ **Phase $3^\top$: Backprop through output materialization (parallel)**

for $k = 1$ **to** M **in parallel do**

  $\bar{\mathbf{O}}_{\text{hist}} \leftarrow \bar{\mathbf{O}}^{(k)}$, $\bar{\mathbf{O}}_{\text{intra}} \leftarrow \bar{\mathbf{O}}^{(k)}$.

▷ History term: $\mathbf{O}_{\text{hist}} = \text{Diag}(\boldsymbol{\delta})(\mathbf{Q}\mathbf{S}_{\text{in}})$

$\mathbf{H} \leftarrow \mathbf{Q}^{(k)}\mathbf{S}_{\text{in}}^{(k)}$ ▷ $C \times d_v$

$\bar{\mathbf{H}} \leftarrow \text{Diag}(\boldsymbol{\delta}^{(k)})\,\bar{\mathbf{O}}_{\text{hist}}$

$\bar{\boldsymbol{\delta}}^{(k)} \mathrel{+}= \text{row\_dot}(\bar{\mathbf{O}}_{\text{hist}}, \mathbf{H})$

$\bar{\mathbf{Q}}^{(k)} \mathrel{+}= \bar{\mathbf{H}}\,(\mathbf{S}_{\text{in}}^{(k)})^\top$

$\bar{\mathbf{S}}_{\text{in, out}}^{(k)} \mathrel{+}= (\mathbf{Q}^{(k)})^\top \bar{\mathbf{H}}$

▷ Intra term: $\mathbf{O}_{\text{intra}} = ((\mathbf{Q}\mathbf{X}_{\text{ext}}^\top) \odot \mathbf{M})\mathbf{V}_{\text{ext}}$

$\mathbf{P} \leftarrow \mathbf{Q}^{(k)}(\mathbf{X}_{\text{ext}}^{(k)})^\top, \quad \mathbf{W} \leftarrow \mathbf{P} \odot \mathbf{M}^{(k)}$

$\bar{\mathbf{W}} \leftarrow \bar{\mathbf{O}}_{\text{intra}}(\mathbf{V}_{\text{ext}}^{(k)})^\top$

$\bar{\mathbf{V}}_{\text{ext}}^{(k)} \mathrel{+}= \mathbf{W}^\top \bar{\mathbf{O}}_{\text{intra}}$

$\bar{\mathbf{P}} \leftarrow \bar{\mathbf{W}} \odot \mathbf{M}^{(k)}$

$\bar{\mathbf{M}}^{(k)} \mathrel{+}= \bar{\mathbf{W}} \odot \mathbf{P}$

$\bar{\mathbf{Q}}^{(k)} \mathrel{+}= \bar{\mathbf{P}}\mathbf{X}_{\text{ext}}^{(k)}$

$\bar{\mathbf{X}}_{\text{ext}}^{(k)} \mathrel{+}= \bar{\mathbf{P}}^\top \mathbf{Q}^{(k)}$

end for

▷ **Phase $2^\top$: Backprop through inter-chunk boundary propagation (sequential reverse)**

$\bar{\mathbf{S}}_{\text{in}}^{(M+1)} \leftarrow \mathbf{0}_{d_x \times d_v}$ ▷ no loss on final state by default

for $k = M$ **down to** 1 **do**

$\bar{\mathbf{S}}_{\text{out}}^{(k)} \leftarrow \bar{\mathbf{S}}_{\text{in}}^{(k+1)}$ ▷ $\mathbf{S}_{\text{in}}^{(k+1)} = \mathbf{S}_{\text{out}}^{(k)}$

$\bar{\mathbf{S}}_{\text{in}}^{(k)} \mathrel{+}= \bar{\mathbf{S}}_{\text{in, out}}^{(k)}$

▷ Chunk recurrence: $\mathbf{S}_{\text{out}}^{(k)} = g^{(k)}\mathbf{S}_{\text{in}}^{(k)} + \mathbf{B}^{(k)}$

$\bar{\mathbf{B}}^{(k)} \leftarrow \bar{\mathbf{S}}_{\text{out}}^{(k)}$

$\bar{g}^{(k)} \leftarrow \langle \bar{\mathbf{S}}_{\text{out}}^{(k)}, \mathbf{S}_{\text{in}}^{(k)} \rangle_F$

$\bar{\mathbf{S}}_{\text{in}}^{(k)} \mathrel{+}= g^{(k)}\,\bar{\mathbf{S}}_{\text{out}}^{(k)}$

▷ Bias: $\mathbf{B}^{(k)} = \mathbf{X}_{\text{ext}}^\top(\text{Diag}(m_{\text{end}})\mathbf{V}_{\text{ext}})$

$m_{\text{end}} \leftarrow \mathbf{M}_{C,:}^{(k)}, \quad \mathbf{W}_v \leftarrow \text{Diag}(m_{\text{end}})\mathbf{V}_{\text{ext}}^{(k)}$

$\bar{\mathbf{X}}_{\text{ext}}^{(k)} \mathrel{+}= \mathbf{W}_v(\bar{\mathbf{B}}^{(k)})^\top$

$\bar{\mathbf{W}}_v \leftarrow \mathbf{X}_{\text{ext}}^{(k)}\bar{\mathbf{B}}^{(k)}$

$\bar{m}_{\text{end}} \leftarrow \text{row\_dot}(\bar{\mathbf{W}}_v, \mathbf{V}_{\text{ext}}^{(k)})$

$\bar{\mathbf{V}}_{\text{ext}}^{(k)} \mathrel{+}= \text{Diag}(m_{\text{end}})\,\bar{\mathbf{W}}_v$

$\bar{\mathbf{M}}_{C,:}^{(k)} \mathrel{+}= \bar{m}_{\text{end}}$

▷ Chunk-exit decay: $g^{(k)} = \delta_C^{(k)}$

$\bar{\boldsymbol{\delta}}_C^{(k)} \mathrel{+}= \bar{g}^{(k)}$

end for

$\bar{\mathbf{S}}_{\text{init}} \leftarrow \bar{\mathbf{S}}_{\text{in}}^{(1)}$

▷ **Phase $1^\top$: Backprop through structured mask and local log-decay (parallel)**

▷ Helper sums: $\text{row\_sum}(\mathbf{G}) = \mathbf{G}\mathbf{1}$ and $\text{col\_sum}(\mathbf{G}) = \mathbf{G}^\top\mathbf{1}$.

for $k = 1$ **to** M **in parallel do**

$a \leftarrow a_k,\ b \leftarrow b_k,\ J \leftarrow J_k.$

▷ Mask factorization: $\mathbf{M} = \mathbf{D}\ \text{Diag}(\boldsymbol{\rho})\,\mathbf{A}$

$\mathbf{E} \leftarrow \text{Diag}(\boldsymbol{\rho}_{a:b})\,\mathbf{A}^{(k)}$

$\bar{\mathbf{D}} \leftarrow \bar{\mathbf{M}}^{(k)}\mathbf{E}^\top, \quad \bar{\mathbf{E}} \leftarrow (\mathbf{D}^{(k)})^\top\bar{\mathbf{M}}^{(k)}$

$\bar{\boldsymbol{\rho}}_{a:b} \mathrel{+}= \text{row\_dot}(\bar{\mathbf{E}}, \mathbf{A}^{(k)})$

▷ Local log-decay: $\delta_i = \exp(u_i)$, $D_{i,s} = \mathbb{I}[s \le i]\exp(u_i - u_s)$, $u_i = u_{i-1} + \log\gamma_{a+i-1}$ with $u_0 = 0$

Initialize $\bar{u}_{0:C} \leftarrow 0$.

$\bar{u}_{1:C} \mathrel{+}= \boldsymbol{\delta}^{(k)} \odot \bar{\boldsymbol{\delta}}^{(k)}$

$\mathbf{G} \leftarrow \text{tril}(\bar{\mathbf{D}} \odot \mathbf{D}^{(k)}, 0)$

$\bar{u}_{1:C}$ += row_sum($\mathbf{G}$) − col_sum($\mathbf{G}$)
Initialize $\overline{\log}\,\gamma_{a:b} \leftarrow \mathbf{0}_C$
for $i = C$ **down to** 1 **do**
  $t \leftarrow a + i - 1$
  $\overline{\log}\,\gamma_t$ += $\bar{u}_i$
  $\bar{u}_{i-1}$ += $\bar{u}_i$
end for
▷ Backprop $\log \gamma_t = \mathrm{log1p}(-\alpha_t)$
for $t = a, \ldots, b$ **do**
  $\bar{\alpha}_t \leftarrow -\exp(-\log \gamma_t)\,\overline{\log}\,\gamma_t$
  $\bar{\boldsymbol{\alpha}}_t$ += $\bar{\alpha}_t$
end for
end for
▷ **Scatter-add overlap gradients back to global arrays (atomic add if parallel).**
for $k = 1$ **to** M **do**
  $\bar{\mathbf{Q}}_{a_k:b_k}$ += $\bar{\mathbf{Q}}^{(k)}$
  $\bar{\mathbf{X}}_{J_k}$ += $\bar{\mathbf{X}}^{(k)}_{\text{ext}}$
  $\bar{\mathbf{V}}_{J_k}$ += $\bar{\mathbf{V}}^{(k)}_{\text{ext}}$
end for
If the forward pass hard-sets $(\rho_1, \alpha_1) \leftarrow (0, 0)$ (as in Algorithm 5), then set $\bar{\rho}_1 \leftarrow 0$ and $\bar{\alpha}_1 \leftarrow 0$.
return $(\bar{\mathbf{Q}}, \bar{\mathbf{X}}, \bar{\mathbf{V}}, \bar{\boldsymbol{\rho}}, \bar{\boldsymbol{\alpha}}, \bar{\mathbf{S}}_{\text{init}})$.

C Parallelizing Delta Network on Sequence Length Dimension

The basic Delta Network update in Section 3 has the form

$$\mathbf{S}_t = (\mathbf{I} - \eta_t \mathbf{k}_{t-1}\mathbf{k}_{t-1}^\top)\mathbf{S}_{t-1} + \eta_t \mathbf{k}_{t-1}\mathbf{v}_t^\top, \tag{C.1}$$

which corresponds to the projector-only case $\lambda_t = 0$ of Eq. (3.3). When ridge regularization / weight decay is enabled, Eq. (3.3) changes the transition to $\mathbf{A}_t = \gamma_t \mathbf{I} - \eta_t \mathbf{k}_{t-1}\mathbf{k}_{t-1}^\top$ with $\gamma_t := 1 - \eta_t \lambda_t$. For $\gamma_t > 0$ one can factor $\mathbf{A}_t = \gamma_t(\mathbf{I} - \mathbf{u}'_t \mathbf{u}'^\top_t)$ where $\mathbf{u}'_t := \sqrt{\eta_t/\gamma_t}\,\mathbf{k}_{t-1}$, and absorb $\prod \gamma_t$ into a cumulative rescaling of boundary states and bias terms. For clarity, the WY derivation below presents the projector-only case $\lambda_t = 0$.

Explicit reduction to the projector-only case (for $\lambda_t > 0$). Because γ_t is a *scalar*, one can reuse the WY machinery unchanged via a simple rescaling. Let $c_0 := 1$ and $c_t := \prod_{r=1}^t \gamma_r$ and define $\widetilde{\mathbf{S}}_t := \mathbf{S}_t/c_t$, $\widetilde{\eta}_t := \eta_t/\gamma_t$, and $\widetilde{\mathbf{v}}_t := \mathbf{v}_t/c_{t-1}$. Then the decayed recurrence is equivalent to the projector-only recurrence

$$\widetilde{\mathbf{S}}_t = (\mathbf{I} - \widetilde{\eta}_t\, \mathbf{k}_{t-1}\mathbf{k}_{t-1}^\top)\widetilde{\mathbf{S}}_{t-1} + \widetilde{\eta}_t\, \mathbf{k}_{t-1}\widetilde{\mathbf{v}}_t^\top, \qquad \mathbf{o}_t = c_t \widetilde{\mathbf{o}}_t.$$

Chunk-local renormalization. Forming c_t (or c_{t-1}^{-1}) over long sequences can underflow/overflow; all chunk-wise kernels therefore apply this rescaling *within each chunk* by resetting the log-prefix decay to 0 at chunk boundaries (see Algorithm 9 and Algorithm 10).

Step-size convention. Throughout this appendix, η_t denotes the actual step size used in the rank-one update (e.g., the NLMS step size from Eq. (3.4)). In the *projector-only* setting treated

here ($\lambda_t = 0$), a gain $\beta_t \in (0, 2)$ corresponds to

$$\eta_t = \frac{\beta_t}{\|\mathbf{k}_{t-1}\|_2^2 + \varepsilon},$$

with the usual convention $\eta_t := 0$ when the denominator vanishes. Hence $\eta_t \geq 0$ and $\sqrt{\eta_t}$ is well-defined.

C.1 Affine Representation and WY Form

We explicitly define the affine structure. Write

$$\mathbf{A}_t := \mathbf{I} - \eta_t \mathbf{k}_{t-1}\mathbf{k}_{t-1}^\top \in \mathbb{R}^{d\times d}, \qquad \mathbf{B}_t := \eta_t \mathbf{k}_{t-1}\mathbf{v}_t^\top \in \mathbb{R}^{d\times d_v}, \tag{C.2}$$

so that the recurrence becomes $\mathbf{S}_t = \mathbf{A}_t\mathbf{S}_{t-1} + \mathbf{B}_t$ (projector-only; $\lambda_t = 0$). Eq. (C.2) reveals a rank-one structure $\mathbf{A}_t = \mathbf{I} - \mathbf{u}_t\mathbf{u}_t^\top$ where $\mathbf{u}_t := \sqrt{\eta_t}\,\mathbf{k}_{t-1}$. Bischof and Van Loan (1987) show that the product of such matrices over a block can be written as a single WY transform (or UT transform). For a chunk of size C, the accumulated transition matrix $\widehat{\mathbf{A}} = \mathbf{A}_{t+C}\cdots\mathbf{A}_{t+1}$ is:

$$\widehat{\mathbf{A}} = \mathbf{I} - \mathbf{U}\mathbf{T}\mathbf{U}^\top, \tag{C.3}$$

where $\mathbf{U} \in \mathbb{R}^{d\times C}$ collects the update vectors $\mathbf{u}$, and $\mathbf{T} \in \mathbb{R}^{C\times C}$ is a triangular mixing matrix.

C.2 Chunk-wise Parallel Algorithm

Similar to Yang et al. (2024b), we have the following three-phase parallel algorithm. Phase 1 computes the local operators in parallel. Phase 2 performs a fast recurrence over chunk boundaries. Phase 3 materializes the outputs by combining the propagated state with a local causal attention mechanism.

Algorithm 7 Parallel DeltaNet Training (Forward, $\lambda_t = 0$)

Require: Shifted keys $\mathbf{x} \in \mathbb{R}^{L\times d}$ (with $\mathbf{x}_t := \mathbf{k}_{t-1}$), queries $\mathbf{Q} \in \mathbb{R}^{L\times d}$, values $\mathbf{V} \in \mathbb{R}^{L\times d_v}$, step sizes $\boldsymbol{\eta} \in \mathbb{R}^L$ with $\eta_t \geq 0$ (so $\sqrt{\eta_t}$ is defined), initial state $\mathbf{S}_{\text{init}} \in \mathbb{R}^{d\times d_v}$ (default $\mathbf{0}$). Chunk size C (assume $C \mid L$). ▷ Projector-only case $\lambda_t = 0$.

Reshape inputs into $M = L/C$ chunks: $\mathbf{U}_{\text{raw}}^{(k)}, \mathbf{Q}^{(k)} \in \mathbb{R}^{d\times C}$ and $\mathbf{V}^{(k)} \in \mathbb{R}^{C\times d_v}$, with $\boldsymbol{\eta}^{(k)} \in \mathbb{R}^C$.
▷ Here $\mathbf{x}_t := \mathbf{k}_{t-1}$ is the shifted key stream.
▷ **Phase 1: Intra-chunk Pre-computation (Parallel)**
for $k = 1$ **to** M **in parallel do**
  $\mathbf{U}^{(k)} \leftarrow \mathbf{U}_{\text{raw}}^{(k)} \cdot \text{diag}(\sqrt{\boldsymbol{\eta}^{(k)}})$ ▷ Scale key-columns
  $\widetilde{\mathbf{V}}^{(k)} \leftarrow \text{diag}(\sqrt{\boldsymbol{\eta}^{(k)}})\, \mathbf{V}^{(k)}$ ▷ Scale value-rows
  $\mathbf{G}^{(k)} \leftarrow {\mathbf{U}^{(k)}}^\top \mathbf{U}^{(k)}$ ▷ Gram Matrix
  $\mathbf{L}^{(k)} \leftarrow \text{tril}(\mathbf{G}^{(k)}, -1) + \mathbf{I}$ ▷ Implicit $\mathbf{T}^{-1}$
  $\mathbf{Z}^{(k)} \leftarrow \text{solve\_triangular}(\mathbf{L}^{(k)}, \widetilde{\mathbf{V}}^{(k)})$ ▷ Local value solve
  $\widehat{\mathbf{B}}^{(k)} \leftarrow \mathbf{U}^{(k)}\mathbf{Z}^{(k)}$ ▷ Accumulated Bias $\widehat{\mathbf{B}}$ for chunk
end for
▷ **Phase 2: Inter-chunk Recurrence (Sequential)**
$\mathbf{S}_{\text{in}}^{(1)} \leftarrow \mathbf{S}_{\text{init}}$
for $k = 1$ **to** M **do**
  $\Gamma^{(k)} \leftarrow \text{solve\_triangular}(\mathbf{L}^{(k)}, {\mathbf{U}^{(k)}}^\top \mathbf{S}_{\text{in}}^{(k)})$ ▷ Project state onto subspace (cache $\Gamma^{(k)}$)
  $\mathbf{S}_{\text{out}}^{(k)} \leftarrow \mathbf{S}_{\text{in}}^{(k)} - \mathbf{U}^{(k)}\Gamma^{(k)}$ ▷ Apply $\widehat{\mathbf{A}}$
  $\mathbf{S}_{\text{in}}^{(k+1)} \leftarrow \mathbf{S}_{\text{out}}^{(k)} + \widehat{\mathbf{B}}^{(k)}$ ▷ Propagate to next chunk
end for
$\mathbf{S}_{\text{final}} \leftarrow \mathbf{S}_{\text{in}}^{(M+1)}$ ▷ Final state after processing length L
▷ **Phase 3: Materialize Outputs (Parallel)**
for $k = 1$ **to** M **in parallel do**
  $\mathbf{H}^{(k)} \leftarrow {\mathbf{Q}^{(k)}}^\top \mathbf{S}_{\text{in}}^{(k)}$ ▷ $C \times d_v$: Query interaction with incoming state
  $\boldsymbol{\Lambda}^{(k)} \leftarrow {\mathbf{Q}^{(k)}}^\top \mathbf{U}^{(k)}$ ▷ $C \times C$: Query-Key interaction matrix
  $\mathbf{M}^{(k)} \leftarrow \text{tril}(\boldsymbol{\Lambda}^{(k)}, 0)$ ▷ Causal scores including diagonal
  $\mathbf{Y}_{\text{hist}}^{(k)} \leftarrow \mathbf{H}^{(k)} - \mathbf{M}^{(k)}\Gamma^{(k)}$ ▷ History contribution (uses cached $\Gamma^{(k)}$)
  $\mathbf{Y}_{\text{intra}}^{(k)} \leftarrow \mathbf{M}^{(k)}\mathbf{Z}^{(k)}$ ▷ Intra-chunk contribution (uses cached $\mathbf{Z}^{(k)}$)
  $\mathbf{O}^{(k)} \leftarrow \mathbf{Y}_{\text{hist}}^{(k)} + \mathbf{Y}_{\text{intra}}^{(k)}$
end for
▷ Equivalently: $\mathbf{O}^{(k)} = \mathbf{H}^{(k)} + \mathbf{M}^{(k)}(\mathbf{Z}^{(k)} - \Gamma^{(k)})$ with $\mathbf{M}^{(k)} = \text{tril}((\mathbf{Q}^{(k)})^\top \mathbf{U}^{(k)}, 0)$.
return $(\mathbf{O}, \mathbf{S}_{\text{final}})$

Details on Phase 3. To materialize outputs efficiently without reconstructing $\mathbf{S}_t$ at every step, we decompose the output into two terms: $\mathbf{y}_t = \mathbf{y}_t^{\text{hist}} + \mathbf{y}_t^{\text{intra}}$. For chunk k, define the causal score matrix $\mathbf{M}^{(k)} := \text{tril}(\mathbf{Q}^{(k)\top}\mathbf{U}^{(k)}, 0) \in \mathbb{R}^{C\times C}$.

- **History Term ($\mathbf{y}_t^{\text{hist}}$):** For projector-only DeltaNet ($\lambda_t = 0$), the within-chunk projector product is already encoded by the unit-lower triangular factor $\mathbf{L}$ and the projected-state solve $\Gamma^{(k)} = \mathbf{L}^{(k)-1}(\mathbf{U}^{(k)\top}\mathbf{S}_{\text{in}}^{(k)})$ computed in Phase 2. The history contribution is $\mathbf{Y}_{\text{hist}}^{(k)} = \mathbf{Q}^{(k)\top}\mathbf{S}_{\text{in}}^{(k)} - \mathbf{M}^{(k)}\Gamma^{(k)}$.

- **Intra-chunk Term ($\mathbf{y}_t^{\text{intra}}$):** Similarly, Phase 1 computes $\mathbf{Z}^{(k)} = \mathbf{L}^{(k)-1}\widetilde{\mathbf{V}}^{(k)}$. The intra-chunk contribution is the causal matrix product $\mathbf{Y}_{\text{intra}}^{(k)} = \mathbf{M}^{(k)}\mathbf{Z}^{(k)}$.

C.3 Parallel DeltaNet Training (Backward)

Algorithm 8 Parallel DeltaNet Training (Backward, $\lambda_t = 0$)

Require: Upstream gradient $\bar{\mathbf{O}} \in \mathbb{R}^{L\times d_v}$ and cached forward tensors from Algorithm 7 (projector-only, $\lambda_t = 0$) for each chunk $k \in \{1,\dots,M\}$: unscaled keys $\mathbf{U}_{\text{raw}}^{(k)} \in \mathbb{R}^{d\times C}$, queries $\mathbf{Q}^{(k)} \in \mathbb{R}^{d\times C}$, values $\mathbf{V}^{(k)} \in \mathbb{R}^{C\times d_v}$, step sizes $\boldsymbol{\eta}^{(k)} \in \mathbb{R}^C$, scaled keys $\mathbf{U}^{(k)} = \mathbf{U}_{\text{raw}}^{(k)}\,\mathrm{diag}(\sqrt{\boldsymbol{\eta}^{(k)}})$, scaled values $\widetilde{\mathbf{V}}^{(k)} = \mathrm{diag}(\sqrt{\boldsymbol{\eta}^{(k)}})\mathbf{V}^{(k)}$, $\mathbf{L}^{(k)} \in \mathbb{R}^{C\times C}$, $\mathbf{Z}^{(k)} = \mathrm{solve\_triangular}(\mathbf{L}^{(k)}, \widetilde{\mathbf{V}}^{(k)})$, $\Gamma^{(k)} = \mathrm{solve\_triangular}(\mathbf{L}^{(k)}, (\mathbf{U}^{(k)})^\top \mathbf{S}_{\text{in}}^{(k)})$, and boundary states $\mathbf{S}_{\text{in}}^{(k)} \in \mathbb{R}^{d\times d_v}$.

Ensure: Gradients $\bar{\mathbf{Q}} \in \mathbb{R}^{L\times d}$, $\bar{\mathbf{U}}_{\text{raw}} \in \mathbb{R}^{L\times d}$, $\bar{\mathbf{V}} \in \mathbb{R}^{L\times d_v}$, $\bar{\boldsymbol{\eta}} \in \mathbb{R}^L$, and $\bar{\mathbf{S}}_{\text{init}} \in \mathbb{R}^{d\times d_v}$.

Reshape $\bar{\mathbf{O}}$ into chunks $\bar{\mathbf{O}}^{(k)} \in \mathbb{R}^{C\times d_v}$, $M = L/C$.

Initialize $\bar{\mathbf{Q}}^{(k)}, \bar{\mathbf{U}}^{(k)}, \bar{\widetilde{\mathbf{V}}}^{(k)}, \bar{\mathbf{L}}^{(k)}, \bar{\mathbf{S}}_{\text{in}}^{(k)}, \bar{\mathbf{Z}}^{(k)}, \bar{\Gamma}^{(k)} \leftarrow \mathbf{0}$ for all k.

Set $\bar{\mathbf{S}}_{\text{in}}^{(M+1)}$ to the upstream gradient on the final chunk-exit state (default $\mathbf{0}$ if unused by the loss).

▷ **Adjoint identity (triangular solve).** Let L be unit-lower triangular with fixed diagonal. If $X = L^{-1}B$,

▷ then $\bar{B} = L^{-T}\bar{X}$ and $\bar{L} \mathrel{-}= \mathrm{tril}(\bar{B}X^\top, -1)$.

▷ **Adjoint identity (Gram).** If $G = U^\top U$, then $\bar{U} \leftarrow \bar{U} + U(\bar{G} + \bar{G}^\top)$.

▷ **Phase $3^\top$: Backprop through output materialization (Parallel)**

for $k = 1$ **to** M **in parallel do**

▷ Projector-only identity: $\mathbf{O}^{(k)} = \mathbf{H}^{(k)} + \mathbf{M}^{(k)}(\mathbf{Z}^{(k)} - \Gamma^{(k)})$ with $\mathbf{H}^{(k)} = (\mathbf{Q}^{(k)})^\top \mathbf{S}_{\text{in}}^{(k)}$ and $\mathbf{M}^{(k)} = \mathrm{tril}((\mathbf{Q}^{(k)})^\top \mathbf{U}^{(k)}, 0)$.

$\bar{\mathbf{H}}^{(k)} \leftarrow \bar{\mathbf{O}}^{(k)}$

$\boldsymbol{\Lambda}^{(k)} \leftarrow (\mathbf{Q}^{(k)})^\top \mathbf{U}^{(k)}$

$\mathbf{M}^{(k)} \leftarrow \mathrm{tril}(\boldsymbol{\Lambda}^{(k)}, 0)$

$\mathbf{E}^{(k)} \leftarrow \mathbf{Z}^{(k)} - \Gamma^{(k)}$ ▷ $C \times d_v$

$\bar{\mathbf{M}}^{(k)} \leftarrow \bar{\mathbf{O}}^{(k)}(\mathbf{E}^{(k)})^\top$

$\bar{\mathbf{M}}^{(k)} \leftarrow \mathrm{tril}(\bar{\mathbf{M}}^{(k)}, 0)$ ▷ $\mathbf{M}^{(k)}$ is causal (includes diagonal)

$\bar{\mathbf{E}}^{(k)} \leftarrow (\mathbf{M}^{(k)})^\top \bar{\mathbf{O}}^{(k)}$

$\bar{\mathbf{Z}}^{(k)} \leftarrow \bar{\mathbf{Z}}^{(k)} + \bar{\mathbf{E}}^{(k)}$

$\bar{\Gamma}^{(k)} \leftarrow \bar{\Gamma}^{(k)} - \bar{\mathbf{E}}^{(k)}$

▷ Backprop $\mathbf{H}^{(k)} = (\mathbf{Q}^{(k)})^\top \mathbf{S}_{\text{in}}^{(k)}$

$\bar{\mathbf{Q}}^{(k)} \leftarrow \bar{\mathbf{Q}}^{(k)} + \mathbf{S}_{\text{in}}^{(k)}(\bar{\mathbf{H}}^{(k)})^\top$

$\bar{\mathbf{S}}_{\text{in}}^{(k)} \leftarrow \bar{\mathbf{S}}_{\text{in}}^{(k)} + \mathbf{Q}^{(k)}\bar{\mathbf{H}}^{(k)}$

▷ Backprop $\mathbf{M}^{(k)} = \mathrm{tril}(\boldsymbol{\Lambda}^{(k)}, 0)$ and $\boldsymbol{\Lambda}^{(k)} = (\mathbf{Q}^{(k)})^\top \mathbf{U}^{(k)}$

$\bar{\boldsymbol{\Lambda}}^{(k)} \leftarrow \bar{\mathbf{M}}^{(k)}$

$\bar{\mathbf{Q}}^{(k)} \leftarrow \bar{\mathbf{Q}}^{(k)} + \mathbf{U}^{(k)}(\bar{\boldsymbol{\Lambda}}^{(k)})^\top$

$\bar{\mathbf{U}}^{(k)} \leftarrow \bar{\mathbf{U}}^{(k)} + \mathbf{Q}^{(k)}\bar{\boldsymbol{\Lambda}}^{(k)}$

end for

▷ **Phase $2^\top$: Backprop through inter-chunk recurrence (Sequential reverse)**

for $k = M$ **down to** 1 **do**

$\bar{\mathbf{S}}_{\text{next}}^{(k)} \leftarrow \bar{\mathbf{S}}_{\text{in}}^{(k+1)}$

▷ Link: $\mathbf{S}_{\text{in}}^{(k+1)} = \mathbf{S}_{\text{out}}^{(k)} + \widehat{\mathbf{B}}^{(k)}$

$\bar{\mathbf{S}}_{\text{out}}^{(k)} \leftarrow \bar{\mathbf{S}}_{\text{next}}^{(k)}$

$\bar{\widehat{\mathbf{B}}}^{(k)} \leftarrow \bar{\mathbf{S}}_{\text{next}}^{(k)}$

▷ Bias: $\widehat{\mathbf{B}}^{(k)} = \mathbf{U}^{(k)}\mathbf{Z}^{(k)}$

$\bar{\mathbf{U}}^{(k)} \leftarrow \bar{\mathbf{U}}^{(k)} + \bar{\tilde{\mathbf{B}}}^{(k)} (\mathbf{Z}^{(k)})^\top$

$\bar{\mathbf{Z}}^{(k)} \leftarrow \bar{\mathbf{Z}}^{(k)} + (\mathbf{U}^{(k)})^\top \bar{\tilde{\mathbf{B}}}^{(k)}$

▷ Solve: $\mathbf{Z}^{(k)} = \text{solve\_triangular}(\mathbf{L}^{(k)}, \widetilde{\mathbf{V}}^{(k)})$

$\bar{\tilde{\mathbf{V}}}^{(k)}_{\text{solve}} \leftarrow \text{solve\_triangular}(\mathbf{L}^{(k)\top}, \bar{\mathbf{Z}}^{(k)})$

$\bar{\tilde{\mathbf{V}}}^{(k)} \leftarrow \bar{\tilde{\mathbf{V}}}^{(k)} + \bar{\tilde{\mathbf{V}}}^{(k)}_{\text{solve}}$

$\bar{\mathbf{L}}^{(k)} \leftarrow \bar{\mathbf{L}}^{(k)} - \text{tril}\big(\bar{\tilde{\mathbf{V}}}^{(k)}_{\text{solve}} (\mathbf{Z}^{(k)})^\top, -1\big)$

▷ State: $\mathbf{S}^{(k)}_{\text{out}} = \mathbf{S}^{(k)}_{\text{in}} - \mathbf{U}^{(k)} \Gamma^{(k)}$

$\bar{\mathbf{S}}^{(k)}_{\text{in}} \leftarrow \bar{\mathbf{S}}^{(k)}_{\text{in}} + \bar{\mathbf{S}}^{(k)}_{\text{out}}$

$\bar{\mathbf{U}}^{(k)} \leftarrow \bar{\mathbf{U}}^{(k)} - \bar{\mathbf{S}}^{(k)}_{\text{out}} (\Gamma^{(k)})^\top$

$\bar{\Gamma}^{(k)} \leftarrow \bar{\Gamma}^{(k)} - (\mathbf{U}^{(k)})^\top \bar{\mathbf{S}}^{(k)}_{\text{out}}$

▷ Solve: $\Gamma^{(k)} = \text{solve\_triangular}(\mathbf{L}^{(k)}, (\mathbf{U}^{(k)})^\top \mathbf{S}^{(k)}_{\text{in}})$

$\bar{\mathbf{P}}^{(k)}_{\text{proj}} \leftarrow \text{solve\_triangular}(\mathbf{L}^{(k)\top}, \bar{\Gamma}^{(k)})$

$\bar{\mathbf{L}}^{(k)} \leftarrow \bar{\mathbf{L}}^{(k)} - \text{tril}\big(\bar{\mathbf{P}}^{(k)}_{\text{proj}} (\Gamma^{(k)})^\top, -1\big)$

▷ Projection input: $\mathbf{P}^{(k)}_{\text{proj}} = (\mathbf{U}^{(k)})^\top \mathbf{S}^{(k)}_{\text{in}}$

$\bar{\mathbf{U}}^{(k)} \leftarrow \bar{\mathbf{U}}^{(k)} + \mathbf{S}^{(k)}_{\text{in}} (\bar{\mathbf{P}}^{(k)}_{\text{proj}})^\top$

$\bar{\mathbf{S}}^{(k)}_{\text{in}} \leftarrow \bar{\mathbf{S}}^{(k)}_{\text{in}} + \mathbf{U}^{(k)} \bar{\mathbf{P}}^{(k)}_{\text{proj}}$

end for

$\bar{\mathbf{S}}_{\text{init}} \leftarrow \bar{\mathbf{S}}^{(1)}_{\text{in}}$

▷ **Phase $\mathbf{1}^\top$: Backprop through local structure and input scaling (Parallel)**

for $k = 1$ **to** M **in parallel do**

$\bar{\mathbf{L}}^{(k)} \leftarrow \text{tril}(\bar{\mathbf{L}}^{(k)}, -1)$ ▷ diag of $\mathbf{L}^{(k)}$ is constant $(= \mathbf{I})$

$\bar{\mathbf{G}}^{(k)} \leftarrow \bar{\mathbf{L}}^{(k)}$ ▷ since $\mathbf{L}^{(k)} = \text{tril}(\mathbf{G}^{(k)}, -1) + \mathbf{I}$

$\bar{\mathbf{U}}^{(k)} \leftarrow \bar{\mathbf{U}}^{(k)} + \mathbf{U}^{(k)} \big(\bar{\mathbf{G}}^{(k)} + (\bar{\mathbf{G}}^{(k)})^\top\big)$ ▷ backprop $\mathbf{G}^{(k)} = (\mathbf{U}^{(k)})^\top \mathbf{U}^{(k)}$

$\boldsymbol{\sigma}^{(k)} \leftarrow \sqrt{\boldsymbol{\eta}^{(k)}}$

▷ Unscale: $\mathbf{U}^{(k)} = \mathbf{U}^{(k)}_{\text{raw}} \,\text{diag}(\boldsymbol{\sigma}^{(k)})$ and $\widetilde{\mathbf{V}}^{(k)} = \text{diag}(\boldsymbol{\sigma}^{(k)}) \mathbf{V}^{(k)}$

$\bar{\mathbf{U}}^{(k)}_{\text{raw}} \leftarrow \bar{\mathbf{U}}^{(k)} \,\text{diag}(\boldsymbol{\sigma}^{(k)})$

$\bar{\mathbf{V}}^{(k)} \leftarrow \text{diag}(\boldsymbol{\sigma}^{(k)}) \bar{\tilde{\mathbf{V}}}^{(k)}$

▷ Column/row dot definitions: $(\text{col\_dot}(A, B))_i = \langle A_{:,i}, B_{:,i} \rangle$, $(\text{row\_dot}(A, B))_i = \langle A_{i,:}, B_{i,:} \rangle$.

$\bar{\boldsymbol{\sigma}}^{(k)} \leftarrow \text{col\_dot}(\mathbf{U}^{(k)}_{\text{raw}}, \bar{\mathbf{U}}^{(k)}) + \text{row\_dot}(\mathbf{V}^{(k)}, \bar{\tilde{\mathbf{V}}}^{(k)})$

$\mathbf{m}^{(k)} \leftarrow \mathbb{I}[\boldsymbol{\sigma}^{(k)} > 0]$ ▷ element-wise mask

$\bar{\boldsymbol{\eta}}^{(k)} \leftarrow \mathbf{m}^{(k)} \odot \big(\bar{\boldsymbol{\sigma}}^{(k)} \,/\, (2\boldsymbol{\sigma}^{(k)})\big)$ ▷ Safe when some $\eta_t = 0$ (e.g., boundary $\mathbf{x}_t = \mathbf{0}$): set $\bar{\eta}_t = 0$ instead of forming $0/0$.

end for

return $\bar{\mathbf{Q}}, \bar{\mathbf{U}}_{\text{raw}}, \bar{\mathbf{V}}, \bar{\boldsymbol{\eta}}, \bar{\mathbf{S}}_{\text{init}}$ (reshape chunk grads back to length L).

D Parallel Implementation of FALCON-2

In this appendix, we detail the efficient parallel implementation of **FALCON-2**. FALCON-2 introduces *column-wise* adaptive learning rates $\boldsymbol{\eta}_t$. This implies that every value channel $j \in \{1, \ldots, d_v\}$ follows a unique trajectory in the state space, seemingly preventing the use of shared block-transition matrices.

However, we observe that while the update learning rate varies per column, the *geometry* of the updates is determined solely by the keys $\mathbf{k}_t$, which are shared across all value channels. By

exploiting this structure, we can vectorize the chunk-wise recurrence without grouping channels into heads, allowing for per-dimension adaptivity with minimal overhead.

D.1 Channel-Independent Vectorized Form

Let $\mathbf{S}_t \in \mathbb{R}^{d \times d_v}$ be the state. The FALCON-2 update for the j-th column $\mathbf{s}_{t,j}$ is:

$$\begin{aligned}\mathbf{s}_{t,j} &= \mathbf{s}_{t-1,j} + \eta_{t,j}\,\mathbf{k}_{t-1}\Big(v_{t,j} - \langle \mathbf{k}_{t-1}, \mathbf{s}_{t-1,j}\rangle\Big) \\ &= \Big(\mathbf{I} - \eta_{t,j}\,\mathbf{k}_{t-1}\mathbf{k}_{t-1}^\top\Big)\mathbf{s}_{t-1,j} + \eta_{t,j}\,v_{t,j}\,\mathbf{k}_{t-1},\end{aligned} \tag{D.1}$$

where we assume $\eta_{t,j} \geq 0$ so that $\sqrt{\eta_{t,j}}$ is well-defined. This makes explicit that the rank-one update matrix $\mathbf{I} - \eta_{t,j}\mathbf{k}_{t-1}\mathbf{k}_{t-1}^\top$ depends on the per-channel step size $\eta_{t,j}$, which determines the chunk-local WY system. To parallelize this over a chunk of size C, we utilize the dual (Gram) formulation.

Projector-only form. For clarity, we present the projector-only case (no explicit per-column decay factor). When ridge/weight decay is enabled in the main text, the j-th column update includes a scalar decay $\gamma_{j,t} := 1 - \lambda_t \eta_{j,t}$:

$$\mathbf{s}_{t,j} = (\gamma_{t,j}\mathbf{I} - \eta_{t,j}\mathbf{k}_{t-1}\mathbf{k}_{t-1}^\top)\mathbf{s}_{t-1,j} + \eta_{t,j}v_{t,j}\mathbf{k}_{t-1}, \qquad \gamma_{t,j} := 1 - \lambda_t\eta_{t,j}.$$

Since each value channel evolves independently, this decay can be removed by a *channel-wise* cumulative rescaling (equivalently, right-multiplying $\mathbf{S}_t$ by a diagonal matrix of inverse cumulative decays), reducing back to the projector-only form; cf. Appendix E for the shared-decay special case.

Shared Gram Matrix. Let $\mathbf{K} \in \mathbb{R}^{d \times C}$ be the matrix of (shifted) write keys in the current chunk. The core correlation structure is given by the Gram matrix $\mathbf{G} = \mathbf{K}^\top\mathbf{K} \in \mathbb{R}^{C \times C}$. Crucially, $\mathbf{G}$ is independent of the value dimension d_v and needs to be computed only once per chunk. For channel-wise step sizes, the actual triangular system differs per channel via a simple element-wise modulation of this shared base Gram.

Batched Triangular Solve. The variations in per-channel step sizes $\eta_{t,j}$ modulate this Gram matrix. For each channel j, the implicit system matrix $\mathbf{L}_j$ for the WY representation is:

$$\mathbf{L}_j = \mathrm{tril}\Big(\mathbf{G} \odot (\sqrt{\boldsymbol{\eta}}_j\sqrt{\boldsymbol{\eta}}_j^\top), -1\Big) + \mathbf{I}_C, \tag{D.2}$$

where $\sqrt{\boldsymbol{\eta}}_j \in \mathbb{R}^C$ is the vector of step-size square-roots for channel j over the chunk, and $\odot$ denotes element-wise multiplication (broadcasting). Equivalently, letting $\mathbf{U}_j := \mathbf{K}\,\mathrm{diag}(\sqrt{\boldsymbol{\eta}}_j)$, we have

$$\mathbf{G}_j := \mathbf{U}_j^\top\mathbf{U}_j = \mathbf{G} \odot (\sqrt{\boldsymbol{\eta}}_j\sqrt{\boldsymbol{\eta}}_j^\top), \qquad \mathbf{L}_j = \mathrm{tril}(\mathbf{G}_j, -1) + \mathbf{I}_C.$$

Because the chunk size C is typically small (e.g., $C = 64$ or 128), we can efficiently perform a batched triangular solve over the batch dimension d_v.

D.2 Algorithm

Algorithm 2 describes the chunk-wise forward pass in a form directly parallel to the scalar-step-size WY kernel in Appendix C, but batched over value channels. The only difference from the shared-step-size case is that the unit-lower-triangular system $\mathbf{L}_j$ is channel-dependent (via $\boldsymbol{\eta}_{:,j}$), so the triangular solves are performed in batch over $j \in \{1, \ldots, d_v\}$.

Chunk read/write formulas. Let $\mathbf{S}_{\text{in}} \in \mathbb{R}^{d\times d_v}$ be the incoming state for a chunk, and let $\mathbf{K}, \mathbf{Q} \in \mathbb{R}^{d\times C}$ and $\mathbf{V} \in \mathbb{R}^{C\times d_v}$ be the chunk keys, queries, and values (with causal read-after-write ordering inside the chunk). Define layout convention: inside a chunk, we use the feature-major (column-major) layout $\mathbf{K}, \mathbf{Q} \in \mathbb{R}^{d\times C}$ with tokens along columns. Concretely, if $\mathbf{K}_{(m)}, \mathbf{Q}_{(m)} \in \mathbb{R}^{C\times d}$ are the standard time-major slices for chunk m, then we set $\mathbf{K} := \mathbf{K}_{(m)}^\top$ and $\mathbf{Q} := \mathbf{Q}_{(m)}^\top$. We keep $\mathbf{V} \in \mathbb{R}^{C\times d_v}$ time-major, and similarly $\boldsymbol{\eta} \in \mathbb{R}^{C\times d_v}$ for per-channel step sizes.

$$\mathbf{G} := \mathbf{K}^\top\mathbf{K} \in \mathbb{R}^{C\times C}, \qquad \mathbf{M} := \operatorname{tril}(\mathbf{Q}^\top\mathbf{K}, 0) \in \mathbb{R}^{C\times C}, \qquad \mathbf{H} := \mathbf{Q}^\top\mathbf{S}_{\text{in}} \in \mathbb{R}^{C\times d_v},$$

and the projected incoming state $\mathbf{P} := \mathbf{K}^\top\mathbf{S}_{\text{in}} \in \mathbb{R}^{C\times d_v}$. With per-channel step sizes $\boldsymbol{\eta} \in \mathbb{R}^{C\times d_v}$ and $\boldsymbol{\Sigma} := \sqrt{\boldsymbol{\eta}}$, define for each channel j the unit-lower-triangular

$$\mathbf{L}_j := \mathbf{I}_C + \operatorname{tril}\big(\mathbf{G} \odot (\boldsymbol{\Sigma}_{:,j}\boldsymbol{\Sigma}_{:,j}^\top), -1\big).$$

Throughout we include the diagonal (read-after-write) via $\operatorname{tril}(\cdot, 0)$.

Unscaled scores. Note that $\mathbf{M}$ is intentionally formed with the *unscaled* keys $\mathbf{K}$. All $\sqrt{\eta}$ factors are absorbed into the coefficient matrix $\mathbf{B}$ below; equivalently, for each channel j, $\mathbf{M}(\boldsymbol{\Sigma}_{:,j} \odot \cdot) = \operatorname{tril}(\mathbf{Q}^\top\mathbf{K}\operatorname{diag}(\boldsymbol{\Sigma}_{:,j}), 0)(\cdot)$.

Then, for each channel j, solve

$$\mathbf{z}_j := \mathbf{L}_j^{-1}(\boldsymbol{\Sigma}_{:,j} \odot \mathbf{V}_{:,j}) \in \mathbb{R}^C, \qquad \boldsymbol{\gamma}_j := \mathbf{L}_j^{-1}(\boldsymbol{\Sigma}_{:,j} \odot \mathbf{P}_{:,j}) \in \mathbb{R}^C.$$

Stacking the per-channel solutions as columns gives matrices $\mathbf{Z}, \boldsymbol{\Gamma} \in \mathbb{R}^{C\times d_v}$ with $\mathbf{Z}_{:,j} = \mathbf{z}_j$ and $\boldsymbol{\Gamma}_{:,j} = \boldsymbol{\gamma}_j$. Define the coefficient matrix $\mathbf{B} \in \mathbb{R}^{C\times d_v}$ by

$$\mathbf{B}_{:,j} := \boldsymbol{\Sigma}_{:,j} \odot (\mathbf{z}_j - \boldsymbol{\gamma}_j), \qquad \text{equivalently} \qquad \mathbf{B} := \boldsymbol{\Sigma} \odot (\mathbf{Z} - \boldsymbol{\Gamma}).$$

The chunk outputs and chunk-exit state are then

$$\mathbf{O} = \mathbf{H} + \mathbf{M}\mathbf{B} \in \mathbb{R}^{C\times d_v}, \qquad \mathbf{S}_{\text{out}} = \mathbf{S}_{\text{in}} + \mathbf{K}\mathbf{B} \in \mathbb{R}^{d\times d_v}.$$

Pseudocode. Algorithm 2 in the main text provides pseudocode for the projector-only ($\lambda_t = 0$) forward pass; this appendix focuses on the derivation, complexity, and the $\lambda_t > 0$ extension.

Efficiency Analysis. Per chunk, forming the shared Gram and score matrices costs $\mathcal{O}(dC^2)$ for $\mathbf{K}^\top\mathbf{K}$ and $\mathbf{Q}^\top\mathbf{K}$. Constructing the channel-specific systems $\{\mathbf{L}_j\}$ and performing the batched triangular solves costs $\mathcal{O}(d_vC^2)$. Materializing the chunk outputs via $\mathbf{M}\mathbf{B}$ also costs $\mathcal{O}(d_vC^2)$, and the chunk-exit state update $\mathbf{K}\mathbf{B}$ costs $\mathcal{O}(d\,d_v\,C)$. Compared to a naive per-channel implementation that would require explicit $d \times d$ operators per value dimension, this confines the rate-dependence to small $C \times C$ systems. Nevertheless, the $\mathcal{O}(d_vC^2)$ terms can be substantial for large d_v, motivating the shared-dynamics Lite variant in Appendix E.

Including ridge/weight decay ($\lambda_t > 0$). Algorithm 2 is written for the projector-only case $\lambda_t = 0$. In the main text, however, FALCON-2 uses $\lambda_t > 0$ by default, which induces a multiplicative shrinkage on each value channel. With per-channel step sizes $\eta_{t,j}$ and scalar ridge λ_t, the j-th column evolves as

$$\mathbf{s}_{t,j} = \big((1 - \lambda_t\eta_{t,j})\mathbf{I} - \eta_{t,j}\mathbf{k}_{t-1}\mathbf{k}_{t-1}^\top\big)\mathbf{s}_{t-1,j} + \eta_{t,j}v_{t,j}\mathbf{k}_{t-1}.$$

Define the per-channel decay $\gamma_{t,j} := 1 - \lambda_t \eta_{t,j}$ and cumulative products $c_{0,j} := 1$, $c_{t,j} := \prod_{s=1}^{t} \gamma_{s,j}$, and $\mathbf{D}_t := \mathrm{Diag}(c_{t,1}, \ldots, c_{t,d_v})$. Then the column-wise rescaled state $\widetilde{\mathbf{S}}_t := \mathbf{S}_t \mathbf{D}_t^{-1}$ satisfies a projector-only recurrence:

$$\widetilde{\mathbf{s}}_{t,j} = (\mathbf{I} - \widetilde{\eta}_{t,j} \mathbf{k}_{t-1} \mathbf{k}_{t-1}^{\top}) \widetilde{\mathbf{s}}_{t-1,j} + \widetilde{\eta}_{t,j} \widetilde{v}_{t,j} \mathbf{k}_{t-1}, \qquad \widetilde{\eta}_{t,j} := \eta_{t,j} / \gamma_{t,j}, \qquad \widetilde{v}_{t,j} := v_{t,j} / c_{t-1,j}.$$

Equivalently, one can run Algorithm 2 unchanged on the rescaled variables $\widetilde{\boldsymbol{\eta}}$ and $\widetilde{\mathbf{V}}$ (working in the $\widetilde{\mathbf{S}}$ domain), and then recover the original outputs and state by $\mathbf{O}_t = \widetilde{\mathbf{O}}_t \mathbf{D}_t$ and $\mathbf{S}_t = \widetilde{\mathbf{S}}_t \mathbf{D}_t$ (i.e., element-wise multiplication by $c_{t,j}$ on each channel).

Log-space note. Compute $\log \gamma_{t,j} = \mathrm{log1p}(-\lambda_t \eta_{t,j})$ in fp32 and clamp $\lambda_t \eta_{t,j} \leq 1 - \varepsilon_\gamma$ so that $\gamma_{t,j} > 0$. In chunk-wise kernels, do *not* form global products $c_{t,j}$ (or $\exp(\pm \zeta_{t,j})$) over the full sequence; instead, reset the log-prefix at each chunk boundary and use only chunk-local prefixes (Algorithm 9).

Algorithm 9 Parallelized FALCON-2 (Chunk-wise Forward, $\lambda_t > 0$)

Require: Shifted keys $\mathbf{K} \in \mathbb{R}^{L\times d}$, queries $\mathbf{Q} \in \mathbb{R}^{L\times d}$, values $\mathbf{V} \in \mathbb{R}^{L\times d_v}$, per-channel step sizes $\boldsymbol{\eta} \in \mathbb{R}^{L\times d_v}$ with $\eta_{t,j} \geq 0$, ridge coefficients $\boldsymbol{\lambda} \in \mathbb{R}^{L}_{>0}$, clamp $\varepsilon_\gamma > 0$, chunk size C (assume $C \mid L$), initial state $\mathbf{S}_{\text{init}}$ (original scale; default $\mathbf{0}$).

Ensure: Outputs $\mathbf{O}$ for the decayed per-channel update with $\gamma_{t,j} = 1 - \lambda_t \eta_{t,j}$ (clamped to be > 0), using chunk-local renormalization.

Partition the length-L axis into $M = L/C$ contiguous chunks and use the same chunk layout conventions as Algorithm 2.
$\mathbf{S}_{\text{in}} \leftarrow \mathbf{S}_{\text{init}}$
for $m = 1, \ldots, M$ **do**
  Slice $\mathbf{K}_{(m)}, \mathbf{Q}_{(m)} \in \mathbb{R}^{C\times d}$, $\mathbf{V}_{(m)} \in \mathbb{R}^{C\times d_v}$, $\boldsymbol{\eta}_{(m)} \in \mathbb{R}^{C\times d_v}$, and $\boldsymbol{\lambda}_{(m)} \in \mathbb{R}^{C}$.
  Form column-major $\mathbf{K} := \mathbf{K}_{(m)}^\top \in \mathbb{R}^{d\times C}$ and $\mathbf{Q} := \mathbf{Q}_{(m)}^\top \in \mathbb{R}^{d\times C}$.
  $\boldsymbol{\alpha} \leftarrow \boldsymbol{\eta}_{(m)} \odot (\boldsymbol{\lambda}_{(m)} \mathbf{1}_{d_v}^\top)$ ▷ $\alpha_{t,j} = \lambda_t \eta_{t,j}$
  $\boldsymbol{\alpha} \leftarrow \min(\boldsymbol{\alpha}, 1 - \varepsilon_\gamma)$ ▷ ensure $\gamma_{t,j} = 1 - \alpha_{t,j} > 0$
  $\log \boldsymbol{\gamma} \leftarrow \mathrm{log1p}(-\boldsymbol{\alpha})$ ▷ $C \times d_v$; fp32 recommended
  $\widehat{\boldsymbol{\eta}} \leftarrow \boldsymbol{\eta}_{(m)} \odot \exp(-\log \boldsymbol{\gamma})$ ▷ $\widehat{\eta}_{t,j} = \eta_{t,j}/\gamma_{t,j}$
  ▷ Chunk-local log-prefix decays (reset at chunk boundary)
  $u_{0,:} \leftarrow \mathbf{0}_{d_v}$
  for $t = 1, \ldots, C$ **do**
    $u_{t,:} \leftarrow u_{t-1,:} + \log \boldsymbol{\gamma}_{t,:}, \quad \delta_{t,:} \leftarrow \exp(u_{t,:})$ ▷ $\delta_{t,j} = \prod_{r=1}^{t} \gamma_{r,j}$; fp32 recommended
  end for
  Rescaled values: $\widehat{\mathbf{V}}_{t,:} \leftarrow \mathbf{V}_{(m),t,:} \odot \exp(-u_{t-1,:})$ for $t = 1, \ldots, C$ ▷ $\widehat{v}_{t,j} = v_{t,j}/\delta_{t-1,j}$
  ▷ Projector-only WY/Gram step on $(\widehat{\mathbf{V}}, \widehat{\boldsymbol{\eta}})$ (same algebra as Algorithm 2)
  $\boldsymbol{\Sigma} \leftarrow \sqrt{\widehat{\boldsymbol{\eta}}}$ ▷ $C \times d_v$
  $\mathbf{G} \leftarrow \mathbf{K}^\top \mathbf{K}, \quad \mathbf{M} \leftarrow \mathrm{tril}(\mathbf{Q}^\top \mathbf{K}, 0)$
  $\mathbf{H} \leftarrow \mathbf{Q}^\top \mathbf{S}_{\text{in}}, \quad \mathbf{P} \leftarrow \mathbf{K}^\top \mathbf{S}_{\text{in}}$
  Build $\mathbf{L}_j = \mathbf{I}_C + \mathrm{tril}\big(\mathbf{G} \odot (\boldsymbol{\Sigma}_{:,j} \boldsymbol{\Sigma}_{:,j}^\top), -1\big)$ for $j = 1, \ldots, d_v$.
  $\mathbf{Z} \leftarrow \mathrm{BatchedSolveTri}\big(\{\mathbf{L}_j\}_{j=1}^{d_v}, \widehat{\mathbf{V}} \odot \boldsymbol{\Sigma}\big)$
  $\boldsymbol{\Gamma} \leftarrow \mathrm{BatchedSolveTri}\big(\{\mathbf{L}_j\}_{j=1}^{d_v}, \mathbf{P} \odot \boldsymbol{\Sigma}\big)$
  $\mathbf{B} \leftarrow \boldsymbol{\Sigma} \odot (\mathbf{Z} - \boldsymbol{\Gamma})$
  $\widehat{\mathbf{O}}^{(m)} \leftarrow \mathbf{H} + \mathbf{M}\mathbf{B}, \quad \widehat{\mathbf{S}}_{\text{out}} \leftarrow \mathbf{S}_{\text{in}} + \mathbf{K}\mathbf{B}$
  ▷ Rescale back to the original (decayed) variables
  $\mathbf{O}_{t,:}^{(m)} \leftarrow \widehat{\mathbf{O}}_{t,:}^{(m)} \odot \delta_{t,:}$ for $t = 1, \ldots, C$
  $\mathbf{S}_{\text{in}} \leftarrow \widehat{\mathbf{S}}_{\text{out}} \, \mathrm{Diag}(\delta_{C,:})$
end for
return concatenated outputs $\mathbf{O}$.

E FALCON-2-Lite: Efficient Shared Dynamics

In Appendix D, we derived the vectorized parallel form for FALCON-2. While theoretically optimal, that formulation requires a unique learning rate trajectory for every value dimension $j \in \{1, \ldots, d_v\}$.

This creates a computational bottleneck: the implicit inverse matrix $\mathbf{T}$ (or the Cholesky factor $\mathbf{L}$) depends on the learning rates. Consequently, the full FALCON-2 requires solving d_v distinct

triangular systems of size $C \times C$ for every chunk. Since d_v (the model width) is typically large (e.g., 4096), this prevents the use of efficient block-matrix multiplications and leads to high memory bandwidth usage.

In this section, we introduce **FALCON-2-Lite**, a hardware-efficient variant that enforces a shared adaptive learning rate across all value channels. This constraint reduces the number of distinct $C \times C$ triangular systems that must be constructed from d_v to 1, enabling a single multi-right-hand-side triangular solve per chunk while keeping the per-step projector dynamics shared across value channels.

Asymptotically, applying the state update still costs $\mathcal{O}(d\, d_v\, C)$ and solving a triangular system with d_v right-hand sides costs $\mathcal{O}(d_v C^2)$. The primary savings come from eliminating the need to *construct and factor* d_v distinct $C \times C$ systems: Lite uses one shared system per chunk and a single high-throughput solve with many right-hand sides.

E.1 Formulation: Scalar vs. Vector Learning Rates

In the full FALCON-2, the state update is driven by a vector rate $\boldsymbol{\eta}_t \in \mathbb{R}^{d_v}$. In the Lite variant, we constrain this to a scalar $\eta_t \in \mathbb{R}$, derived from the global energy of the input feature $\mathbf{k}_{t-1}$:

$$\eta_t = \frac{\beta_t}{\|\mathbf{k}_{t-1}\|_2^2 + \lambda_t + \varepsilon}, \qquad \beta_t \in (0, 2),\ \lambda_t \geq 0,\ \varepsilon \geq 0, \tag{E.1}$$

where β_t is a learnable scalar NLMS gain, λ_t is the ridge coefficient (default $\lambda_t > 0$), and ε is a small stabilizer (cf. Eq. (3.4)). The autoregressive update rule for the state matrix $\mathbf{S}_t \in \mathbb{R}^{d \times d_v}$ simplifies to:

$$\mathbf{S}_t = \underbrace{\left((1 - \eta_t \lambda_t)\mathbf{I} - \eta_t \mathbf{k}_{t-1}\mathbf{k}_{t-1}^\top\right)}_{\mathbf{A}_t} \mathbf{S}_{t-1} + \eta_t\, \mathbf{k}_{t-1}\mathbf{v}_t^\top, \qquad \gamma_t := 1 - \eta_t \lambda_t. \tag{E.2}$$

Crucially, the transition matrix $\mathbf{A}_t$ is now *shared* across all columns of $\mathbf{S}_t$. This implies that while different features store different contents (values), they share the same dynamics (write/forget speeds).

Including weight decay. FALCON-2 uses $\lambda_t > 0$ by default, inducing the shrinkage factor $\gamma_t := 1 - \eta_t \lambda_t$. For the log-space reduction below, we require $\gamma_t > 0$ (equivalently, $\eta_t \lambda_t < 1$). Under our default RMSNorm scaling (so $\|\mathbf{k}_{t-1}\|_2^2 \approx d$) and $\lambda_t \in [0, 1]$, Eq. (E.1) implies $\eta_t \lambda_t \lesssim \beta_t/(d+1) < 1$ and therefore $\gamma_t \in (0, 1]$ automatically. More generally (and in finite precision), we clamp the decay fraction $\alpha_t := \eta_t \lambda_t$ as $\alpha_t \leftarrow \min(\alpha_t, 1 - \varepsilon_\gamma)$, i.e. $\gamma_t \leftarrow \max(1 - \eta_t \lambda_t, \varepsilon_\gamma)$, so that $\gamma_t \geq \varepsilon_\gamma > 0$. For the simple boundary convention $\mathbf{k}_0 = \mathbf{0}$ (hence $\mathbf{x}_1 = \mathbf{0}$), we take $\eta_1 = 0$ and $\gamma_1 = 1$ (no write/decay), matching the main-text convention that updates start at $t = 2$. Because γ_t is a scalar, one can reduce the decayed recurrence to the projector-only case by rescaling. Let $c_0 := 1$ and $c_t := \prod_{r=1}^{t} \gamma_r$, and define $\widetilde{\mathbf{S}}_t := \mathbf{S}_t / c_t$ (so under read-after-write, reads satisfy $\mathbf{o}_t = c_t \widetilde{\mathbf{o}}_t$). Then the decayed update is equivalent to running the projector-only WY kernels on

$$\widetilde{\eta}_t := \eta_t / \gamma_t, \qquad \widetilde{\mathbf{v}}_t := \mathbf{v}_t / c_{t-1},$$

and rescaling outputs by c_t. Algorithms 10-11 implement this reduction explicitly.

Numerical stability. The algebraic reduction to the projector-only WY kernels rescales targets by the *inverse* cumulative decay, $\widetilde{\mathbf{v}}_t = \mathbf{v}_t / c_{t-1}$ with $c_t = \prod_{r \leq t} \gamma_r$. While correct in exact arithmetic, in finite

precision c_t may become extremely small when γ_t is noticeably below 1, making $c_{t-1}^{-1} = \exp(-\zeta_{t-1})$ overflow and yielding $\infty \times 0$ patterns (and NaNs) when rescaling back by $\exp(\zeta_t)$.

Stable chunk-local renormalization. To avoid ever forming $\exp(\pm\zeta_t)$ over the full sequence, our implementation performs the same reduction *within each WY chunk* of length C. For a chunk $[a, b]$ (where $b = a+C-1$), define local log-prefix decays $u_0 := 0$ and $u_i := \sum_{r=0}^{i-1} \log \gamma_{a+r}$ for $i = 1, \ldots, C$, with $\delta_i := \exp(u_i)$. We run the projector-only WY kernel on the rescaled values $\widehat{\mathbf{v}}_{a+i-1} := \mathbf{v}_{a+i-1} \exp(-u_{i-1}) = \mathbf{v}_{a+i-1}/\delta_{i-1}$ and step sizes $\widehat{\eta}_t := \eta_t/\gamma_t$ (compute $\log \gamma_t = \mathrm{log1p}(-\alpha_t)$ in fp32 with $\alpha_t := \min(\eta_t \lambda_t, 1 - \varepsilon_\gamma)$, then $\widehat{\eta}_t = \eta_t \exp(-\log \gamma_t)$). Outputs and the chunk-exit state are rescaled back by the *forward* local decays: $\mathbf{o}_{a+i-1} = \delta_i \widehat{\mathbf{o}}_{a+i-1}$ and $\mathbf{S}_b = \delta_C \widehat{\mathbf{S}}_b$. This is algebraically equivalent to the global reduction but bounds exponentials by $O(C)$ rather than $O(L)$.

E.2 Parallel Implementation via Shared WY Decomposition

FALCON-2-Lite uses a *single scalar* step size η_t shared across all value channels. Consequently, within each chunk, the WY factors are shared across columns of $\mathbf{S}$: the per-chunk system matrix $\mathbf{L}^{(k)}$ is *identical* for all d_v columns. This is precisely the setting of Appendix C (projector-only, $\lambda_t = 0$), where the WY implementation yields one unit-lower-triangular matrix $\mathbf{L}^{(k)} \in \mathbb{R}^{C \times C}$ per chunk and all triangular solves are standard *multi-RHS* solves with d_v right-hand sides.

Forward/backward reuse. With shared dynamics, the chunk-parallel attention forward pass is exactly Algorithm 7 and the backward pass through the WY machinery is Algorithm 8. The only additional ingredient in FALCON-2-Lite is that the step size η_t is not a free input: it is produced by the NLMS normalization

$$\eta_t = \frac{\beta_t}{\|\mathbf{x}_t\|_2^2 + \lambda_t + \varepsilon}, \qquad \mathbf{x}_t := \mathbf{k}_{t-1}, \tag{E.3}$$

and the default $\lambda_t > 0$ introduces the additional scalar pathway $\gamma_t = 1 - \eta_t \lambda_t$. Backpropagation therefore, includes chain-rule steps through both η_t and the cumulative decay c_t (Algorithm 11).

E.3 Algorithm

Forward. Compute $(\eta_t, \log \gamma_t)$ (with clamping) and run the projector-only WY kernel on *chunk-locally* renormalized inputs: within each chunk compute local prefixes u_i and $\delta_i = \exp(u_i)$, use $\widehat{\eta}_t = \eta_t/\gamma_t$ and $\widehat{\mathbf{v}}_t = \mathbf{v}_t/\delta_{i-1}$ inside that chunk, and rescale outputs/state by δ_i (Algorithm 10).

Algorithm 10 FALCON-2-Lite (Forward)

Require: Shifted keys $\mathbf{x} \in \mathbb{R}^{L\times d}$ (boundary $\mathbf{x}_1 = \mathbf{0}$), queries $\mathbf{Q} \in \mathbb{R}^{L\times d}$, values $\mathbf{V} \in \mathbb{R}^{L\times d_v}$, gains $\beta \in (0,2)^L$, ridge $\lambda \in \mathbb{R}^L_{\geq 0}$, stabilizer $\varepsilon > 0$, clamp $\varepsilon_\gamma > 0$, dummy $\varepsilon_\eta > 0$, chunk size C (assume $C \mid L$), initial state $\mathbf{S}_{\text{init}}$.

Ensure: Outputs $\mathbf{O}$ for the decayed recurrence with $\eta_t = \beta_t/(\|\mathbf{x}_t\|_2^2 + \lambda_t + \varepsilon)$ and $\gamma_t = 1 - \eta_t\lambda_t$ (clamped).

Compute $\eta_1 \leftarrow 0$, and for $t \geq 2$: $d_t \leftarrow \|\mathbf{x}_t\|_2^2 + \lambda_t + \varepsilon$, $\eta_t \leftarrow \beta_t/d_t$ (fp32 for norms/ratios).
$\alpha_t^{\text{raw}} \leftarrow \eta_t\lambda_t$, $\alpha_t \leftarrow \min(\alpha_t^{\text{raw}}, 1-\varepsilon_\gamma)$ for $t \geq 2$.
$\log\gamma_1 \leftarrow 0$ and for $t \geq 2$: $\log\gamma_t \leftarrow \operatorname{log1p}(-\alpha_t)$ (fp32), $\gamma_t \leftarrow \exp(\log\gamma_t)$.
$\widehat{\eta}_1 \leftarrow \varepsilon_\eta$ (dummy; no effect since $\mathbf{x}_1 = \mathbf{0}$), and for $t \geq 2$: $\widehat{\eta}_t \leftarrow \eta_t/\gamma_t = \eta_t \exp(-\log\gamma_t)$.
Partition into $M = L/C$ chunks $[a_k, b_k] = [(k-1)C+1, kC]$. Set $\mathbf{S}_{\text{in}} \leftarrow \mathbf{S}_{\text{init}}$.
for $k = 1, \ldots, M$ **do**
  $a \leftarrow a_k$, $b \leftarrow b_k$.
  $u_0 \leftarrow 0$ ▷ Local log-prefix decays inside the chunk
  for $i = 1, \ldots, C$ **do**
    $u_i \leftarrow u_{i-1} + \log\gamma_{a+i-1}, \quad \delta_i \leftarrow \exp(u_i)$ ▷ fp32
  end for
  $\widehat{\mathbf{V}}_{a+i-1} \leftarrow \mathbf{V}_{a+i-1}\exp(-u_{i-1})$ for $i = 1, \ldots, C$ ▷ $\widehat{\mathbf{v}} = \mathbf{v}/\delta_{i-1}$
  Run the projector-only WY kernel on the length-C sequence $(\mathbf{x}_{a:b}, \mathbf{Q}_{a:b}, \widehat{\mathbf{V}}_{a:b}, \widehat{\eta}_{a:b})$ with init $\mathbf{S}_{\text{in}}$ (Algorithm 7 with $L = C$), producing $(\widehat{\mathbf{O}}_{a:b}, \widehat{\mathbf{S}}_{\text{out}})$.
  $\mathbf{O}_{a+i-1} \leftarrow \delta_i\,\widehat{\mathbf{O}}_{a+i-1}$ for $i = 1, \ldots, C$.
  $\mathbf{S}_{\text{in}} \leftarrow \delta_C\,\widehat{\mathbf{S}}_{\text{out}}$ ▷ Propagate original chunk-exit state
end for
return $\mathbf{O}$.

Backward. Algorithm 8 provides gradients treating $(\mathbf{x}, \boldsymbol{\eta})$ as independent. FALCON-2-Lite additionally backpropagates through the NLMS map $\eta_t = \beta_t/(\|\mathbf{x}_t\|_2^2 + \lambda_t + \varepsilon)$ and through the *chunk-local* log-decay renormalization (the local prefixes u_i and δ_i) used to handle γ_t (Algorithm 11).

Algorithm 11 FALCON-2-Lite (Backward)

Require: Upstream gradient $\bar{\mathbf{O}}$, chunk-local caches from Algorithm 10 for each chunk k: local prefixes $\{u_i, \delta_i\}_{i=0}^{C}$, $\log\gamma_{a_k:b_k}$, (η, β, λ), clamp mask $m_t = \mathbb{I}[\eta_t\lambda_t < 1 - \varepsilon_\gamma]$, and WY caches from running the projector-only kernel on $(\mathbf{x}_{a_k:b_k}, \mathbf{Q}_{a_k:b_k}, \widehat{\mathbf{V}}_{a_k:b_k}, \widehat{\eta}_{a_k:b_k})$.

Ensure: Gradients $(\bar{\mathbf{Q}}, \bar{\mathbf{x}}, \bar{\mathbf{V}}, \bar{\beta}, \bar{\lambda}, \bar{\mathbf{S}}_{\text{init}})$.

Initialize all gradients to zero. Set boundary $\bar{\mathbf{S}}_{\text{in}}^{(M+1)} \leftarrow \mathbf{0}$.
for $k = M, \ldots, 1$ **do** ▷ reverse over chunks
  $a \leftarrow a_k$, $b \leftarrow b_k$, and let $i = t - a + 1 \in \{1, \ldots, C\}$ be the local index.
  Initialize $\bar{u}_i \leftarrow 0$ for $i = 0, \ldots, C$, $\log\bar{\gamma}_t \leftarrow 0$ and $\bar{\eta}_t \leftarrow 0$ for $t = a, \ldots, b$.
  ▷ Unscale outputs: $\mathbf{O}_t = \delta_i\,\widehat{\mathbf{O}}_t$
  for $t = a, \ldots, b$ **do**
    $i \leftarrow t - a + 1$
    $\bar{\widehat{\mathbf{O}}}_t \leftarrow \delta_i\,\bar{\mathbf{O}}_t$
    $\bar{u}_i \mathrel{+}= \langle \bar{\widehat{\mathbf{O}}}_t, \widehat{\mathbf{O}}_t \rangle$ ▷ $\partial\delta_i/\partial u_i = \delta_i$
  end for
  ▷ Boundary: next chunk sees $\mathbf{S}_{\text{in}}^{(k+1)} = \delta_C\,\widehat{\mathbf{S}}_{\text{out}}$

$\bar{\widehat{\mathbf{S}}}_{\text{out}} \leftarrow \delta_C \, \bar{\mathbf{S}}_{\text{in}}^{(k+1)}$

$\bar{u}_C \mathrel{+}= \langle \bar{\widehat{\mathbf{S}}}_{\text{out}}, \widehat{\mathbf{S}}_{\text{out}} \rangle$

Run the projector-only WY backward on this length-C chunk (Algorithm 8 with $L = C$), using upstream $(\bar{\widehat{\mathbf{O}}}_{a:b}, \bar{\widehat{\mathbf{S}}}_{\text{out}})$, to obtain $(\bar{\mathbf{Q}}_{a:b}, \bar{\mathbf{x}}_{a:b}, \bar{\widehat{\mathbf{V}}}_{a:b}, \bar{\widehat{\eta}}_{a:b}, \bar{\mathbf{S}}_{\text{in}}^{(k)})$.

▷ Unscale values: $\widehat{\mathbf{V}}_t = \mathbf{V}_t \exp(-u_{i-1})$

for $t = a, \ldots, b$ **do**

$i \leftarrow t - a + 1$

$s \leftarrow \exp(-u_{i-1})$

$\bar{\mathbf{V}}_t \mathrel{+}= s \, \bar{\widehat{\mathbf{V}}}_t$

$\bar{u}_{i-1} \mathrel{+}= -\langle s \, \bar{\widehat{\mathbf{V}}}_t, \mathbf{V}_t \rangle$

end for

▷ Unscale step sizes: $\widehat{\eta}_t = \eta_t \exp(-\log \gamma_t)$

for $t = \max(a, 2), \ldots, b$ **do**

$g^{-1} \leftarrow \exp(-\log \gamma_t)$

$\bar{\eta}_t \mathrel{+}= g^{-1} \, \bar{\widehat{\eta}}_t$

$\overline{\log \gamma}_t \mathrel{+}= -(\eta_t \, g^{-1}) \, \bar{\widehat{\eta}}_t$

end for

▷ Backprop local prefix sums: $u_i = u_{i-1} + \log \gamma_{a+i-1}$

for $i = C$ **down to** 1 **do**

$t \leftarrow a + i - 1$

if $t \geq 2$ **then**

$\overline{\log \gamma}_t \mathrel{+}= \bar{u}_i$

end if

$\bar{u}_{i-1} \mathrel{+}= \bar{u}_i$

end for

▷ Backprop $\log \gamma_t = \mathrm{log1p}(-\alpha_t)$ with $\alpha_t = \min(\eta_t \lambda_t, 1 - \varepsilon_\gamma)$

for $t = \max(a, 2), \ldots, b$ **do**

$\bar{\alpha}_t \leftarrow -\exp(-\log \gamma_t) \, \overline{\log \gamma}_t$

$\bar{\alpha}_t \leftarrow m_t \, \bar{\alpha}_t$ ▷ zero gradient when clamped

$\bar{\eta}_t \mathrel{+}= \lambda_t \, \bar{\alpha}_t$

$\bar{\lambda}_t \mathrel{+}= \eta_t \, \bar{\alpha}_t$

end for

▷ Backprop $\eta_t = \beta_t / (\|\mathbf{x}_t\|_2^2 + \lambda_t + \varepsilon)$

for $t = \max(a, 2), \ldots, b$ **do**

$d_t \leftarrow \|\mathbf{x}_t\|_2^2 + \lambda_t + \varepsilon$

$\bar{\beta}_t \mathrel{+}= \bar{\eta}_t / d_t$

$\bar{d}_t \leftarrow -\beta_t \, \bar{\eta}_t / d_t^2$

$\bar{\mathbf{x}}_t \mathrel{+}= 2 \, \bar{d}_t \, \mathbf{x}_t$

$\bar{\lambda}_t \mathrel{+}= \bar{d}_t$

end for

▷ Boundary constants $\widehat{\eta}_1 := \varepsilon_\eta$ and $\log \gamma_1 := 0$ carry no gradient.

end for

return $(\bar{\mathbf{Q}}, \bar{\mathbf{x}}, \bar{\mathbf{V}}, \bar{\beta}, \bar{\lambda}, \bar{\mathbf{S}}_{\text{init}})$ with $\bar{\mathbf{S}}_{\text{init}} = \bar{\mathbf{S}}_{\text{in}}^{(1)}$.

E.4 Complexity Analysis

Table 4 summarizes the per-chunk costs. Sharing the dynamics eliminates the *rate-dependent system construction* overhead that scales as $\mathcal{O}(d_v C^2)$ in the full FALCON-2 (building/factorizing d_v distinct $C \times C$ triangular systems). The remaining triangular solve still scales as $\mathcal{O}(d_v C^2)$ because the state

has d_v columns, but it is a single highly optimized multi-RHS solve, which is substantially more GPU-friendly in practice.

Table 4 Complexity per Chunk (Chunk size C, feature dim d, value dim d_v).

Component	**FALCON-2 (Full)**	**FALCON-2-Lite**
Gram Matrix	$\mathcal{O}(dC^2)$	$\mathcal{O}(dC^2)$
Rate-dependent system build ($\mathbf{L}$)	$\mathcal{O}(\mathbf{d_v C^2})$	$\mathcal{O}(\mathbf{C^2})$
Triangular solve ($\mathbf{L}^{-1}$ with d_v RHS)	$\mathcal{O}(d_v C^2)$	$\mathcal{O}(d_v C^2)$
State Update Projection	$\mathcal{O}(dd_v C)$	$\mathcal{O}(dd_v C)$

FALCON-2-Lite removes the need to construct and factor d_v distinct $C \times C$ systems by enforcing shared dynamics. The remaining triangular solve still scales as $\mathcal{O}(d_v C^2)$, but it becomes a single high-throughput multi-RHS solve, which is substantially more GPU-friendly in practice.

F Efficient Training of FALCON-3 via ParallelFlow

While the SSD framework used in Mamba-2 provides an efficient algorithm for rank-1 updates, **FALCON-3** (Sliding Regression) introduces a dependency on a history window of size B. This effectively transforms the system into a rank-B recurrence. To train this efficiently without resorting to slow materialization of the state matrix or naive sequential processing, we adopt the ParallelFlow framework (Cirone and Salvi, 2025).

In this appendix, we first provide a self-contained introduction to ParallelFlow, treating Linear Attention as a Matrix-Valued Controlled Differential Equation (CDE). We then explicitly map the FALCON-3 update rule to this formalism, demonstrating that it constitutes a specific instance of a Low-Rank Delta Rule solvable via the `tensorInv` algorithm.

F.1 Background: Matrix-Valued CDEs and Low-Rank Drivers

Standard linear recurrences can be viewed as discretizations of a continuous-time process. Let $\mathbf{S}_t \in \mathbb{R}^{d \times d_v}$ be the hidden state. We adopt the left-multiplicative convention and model its evolution as a matrix-valued CDE driven by $\boldsymbol{\omega}$ and $\boldsymbol{\xi}$:

$$d\mathbf{S}_t = d\boldsymbol{\omega}_t \mathbf{S}_t + d\boldsymbol{\xi}_t, \tag{F.1}$$

where $\boldsymbol{\omega}_t \in \mathbb{R}^{d \times d}$ and $\boldsymbol{\xi}_t \in \mathbb{R}^{d \times d_v}$ are matrix-valued paths. The solution over an interval $[s, t]$ factors through a linear propagator (flow) $\mathbf{P}_{t \leftarrow s}$:

$$\mathbf{S}_t = \mathbf{P}_{t \leftarrow s} \mathbf{S}_s + \int_s^t \mathbf{P}_{t \leftarrow r} \, d\boldsymbol{\xi}_r, \quad \text{where } \mathbf{P}_{t \leftarrow s} = \mathbf{I} + \int_s^t d\boldsymbol{\omega}_r \mathbf{P}_{r \leftarrow s}. \tag{F.2}$$

This decomposition separates the temporal dynamics ($\mathbf{P}$) from the computation. ParallelFlow parallelizes this by partitioning the sequence into chunks $[t_{k-1}, t_k]$. The system computes local propagators and accumulated input injections for each chunk in parallel, then links them via a global associative scan.

Transpose-equivalent form. ParallelFlow is commonly presented in the shared-right-factor form $d\boldsymbol{\omega}_t = \mathbf{A}_t\mathbf{B}_t^\top dt,\ d\boldsymbol{\xi}_t = \widetilde{\mathbf{A}}_t\mathbf{B}_t^\top dt$, whereas FALCON-3 is most naturally written, under our left-multiplicative recurrence, in the transpose-equivalent shared-left-factor form

$$\mathbf{S}_{t+1} = \mathbf{S}_t + \mathbf{A}_t\mathbf{B}_t^\top\mathbf{S}_t + \mathbf{A}_t\widetilde{\mathbf{B}}_t^\top.$$

The two views are algebraically equivalent by transposing the low-rank factors/state-update identities. We use the shared-left-factor form throughout this appendix because the sliding regression update naturally shares the window matrix of write features.

Low-Rank Drivers. The core difficulty lies in computing the propagator $\mathbf{P}_{s\to t}$, which is typically an expensive matrix ODE. However, efficient computation is possible if the drivers possess a low-rank structure. We assume rank-R drivers with a shared left factor:

$$d\boldsymbol{\omega}_t = \vec{\mathbf{A}}_t\vec{\mathbf{B}}_t^\top dt, \qquad d\boldsymbol{\xi}_t = \vec{\mathbf{A}}_t\widetilde{\vec{\mathbf{B}}}_t^\top dt, \tag{F.3}$$

with $\vec{\mathbf{A}}_t, \vec{\mathbf{B}}_t \in \mathbb{R}^{d\times R}$ and $\widetilde{\vec{B}}_t \in \mathbb{R}^{d_v\times R}$. This structure allows for an efficiently computable, compact representation of the flow. For discrete tokens, under a forward-Euler discretization (with the step size absorbed into $\vec{B}, \widetilde{\vec{B}}$, equivalently $\Delta t = 1$), the update becomes:

$$\mathbf{S}_{t_{k+1}} = \mathbf{S}_{t_k} + \vec{\mathbf{A}}_{t_k}\vec{\mathbf{B}}_{t_k}^\top\mathbf{S}_{t_k} + \vec{\mathbf{A}}_{t_k}\widetilde{\vec{\mathbf{B}}}_{t_k}^\top. \tag{F.4}$$

The `tensorInv` Algorithm. Solving this low-rank system over a chunk of length L_c can be reduced to a Triangular Tensor Inversion. Let $\mathcal{C} \in \mathbb{R}^{(L_c\times R)\times(L_c\times R)}$ be a tensor representing causal interactions between rank-components. For indices $s, t \in \{1, \dots, L_c\}$ and ranks $i, j \in \{1, \dots, R\}$:

$$[\mathcal{C}]_{t,i}^{s,j} = \delta_{s,t}\delta_{i,j} - \mathbb{I}(s < t)\left[\vec{\mathbf{B}}_t^\top\vec{\mathbf{A}}_s\right]_{i,j}. \tag{F.5}$$

Cirone and Salvi (2025) shows that the propagator can be computed by applying the structured inverse tensor $\mathcal{D} = \mathcal{C}^{-1}$ under the tensor-contraction product, thereby avoiding explicit dense $d_x \times d_x$ chunk propagators. In our setting, the computation is governed by an $(L_cR) \times (L_cR)$ structured causal solve together with matrix multiplications involving d_x and d_v; the compressed asymptotic $\mathcal{O}(L_c^2R + d)$ is therefore misleading here because it hides the dominant dimension-dependent terms.

`tensorInv` outputs (the objects used in Algorithm 4). In practice, we do not materialize the full inverse tensor $\mathcal{D}$. Instead, ParallelFlow applies $\mathcal{C}^{-1}$ to two right-hand sides corresponding to the low-rank drivers. For a chunk of length L_c and rank R, stack (flatten time×rank) the discrete drivers as

$$\vec{\mathbf{A}} \in \mathbb{R}^{d_x\times(L_cR)}, \qquad \vec{\mathbf{B}} \in \mathbb{R}^{d_x\times(L_cR)}, \qquad \widetilde{\vec{B}} \in \mathbb{R}^{d_v\times(L_cR)}.$$

Let $\vec{M} \in \{0,1\}^{(L_cR)\times(L_cR)}$ denote the strictly-causal *block* mask that is 1 for interactions from earlier times $s < t$ and 0 otherwise; equivalently, in (t, i) indexing it is exactly the indicator $\mathbb{I}(s < t)$ used in the definition of $\mathcal{C}$ above. In particular, after flattening time×rank, *same-time* rank components do not interact inside the triangular solve. Then the chunk-local triangular system can be written compactly as

$$\mathcal{C} = \mathbf{Id} - \vec{M} \odot (\vec{B}^\top\vec{A}),$$

and the two solves returned by `tensorInv` are

$$\vec{W} := \mathcal{C}^{-1}\vec{B}^{\top} \in \mathbb{R}^{(L_c R)\times d_x}, \qquad \vec{U} := \mathcal{C}^{-1}\widetilde{\vec{B}}^{\top} \in \mathbb{R}^{(L_c R)\times d_v}. \tag{F.6}$$

We denote these solutions by

$$(\vec{W}, \vec{U}) = \texttt{tensorInv}(\vec{A}, \vec{B}, \widetilde{\vec{B}}).$$

Given an incoming chunk-boundary state $\mathbf{S}_{\text{in}} \in \mathbb{R}^{d_x \times d_v}$, define $\vec{Z} := \vec{U} + \vec{W}\mathbf{S}_{\text{in}} \in \mathbb{R}^{(L_c R)\times d_v}$. Then the chunk-exit state is

$$\mathbf{S}_{\text{out}} = \mathbf{S}_{\text{in}} + \vec{A}\vec{Z}, \tag{F.7}$$

which is the form used by Algorithm 4 (with the additional scalar decay γ_t handled there via chunk-local renormalization).

F.2 Mapping FALCON-3 to ParallelFlow

We now explicitly map the **FALCON-3** (Sliding Regression) update rule derived in Section 4 to this low-rank CDE framework.

The FALCON-3 Recurrence. Recall the mini-batch update with window size B (ignoring the scalar decay $\gamma_t = 1 - \eta_t\lambda_t$, which our chunked implementations handle via the same chunk-local log-decay renormalization used throughout):

$$\begin{aligned}\mathbf{S}_t &= \underbrace{\left(\mathbf{I} - \eta_t \bar{\mathbf{C}}_t^{(B)}\right)}_{\text{Transition Matrix}} \mathbf{S}_{t-1} + \underbrace{\eta_t \bar{\mathbf{N}}_t^{(B)}}_{\text{Input Injection}} \\ &= \left(\mathbf{I} - \frac{\eta_t}{B_t}\sum_{j\in\mathcal{I}_t} \mathbf{k}_{j-1}\mathbf{k}_{j-1}^{\top}\right)\mathbf{S}_{t-1} + \frac{\eta_t}{B_t}\sum_{j\in\mathcal{I}_t}\mathbf{k}_{j-1}\mathbf{v}_j^{\top},\end{aligned} \tag{F.8}$$

where $B_t := |\mathcal{I}_t| \le B$, $\bar{\mathbf{C}}_t^{(B)} := \mathbf{C}_t^{(B)}/B_t$, and $\bar{\mathbf{N}}_t^{(B)} := \mathbf{N}_t^{(B)}/B_t$.

As in Section 3, one may write this in the feature space by defining $\mathbf{x}_j := \phi(\mathbf{k}_{j-1})$ and replacing $\mathbf{k}_{j-1}$ by $\mathbf{x}_j$ throughout, and the unkernelized case has ϕ the identity.

Identification of Drivers. To map this to Eq. (F.3) without boundary ambiguities, we define window matrices using the active index set $\mathcal{I}_t = \{j \mid \max(2, t-B+1) \le j \le t\}$. Let $B_t := |\mathcal{I}_t| \le B$ and order the elements of $\mathcal{I}_t$ increasingly. Define the Windowed Key/Value matrices by stacking only valid indices:

$$\vec{K}_t := \begin{bmatrix}\mathbf{k}_{j_1-1} & \mathbf{k}_{j_2-1} & \dots & \mathbf{k}_{j_{B_t}-1}\end{bmatrix} \in \mathbb{R}^{d\times B_t}, \tag{F.9}$$

$$\vec{V}_t := \begin{bmatrix}\mathbf{v}_{j_1} & \mathbf{v}_{j_2} & \dots & \mathbf{v}_{j_{B_t}}\end{bmatrix} \in \mathbb{R}^{d_v\times B_t}, \tag{F.10}$$

where $(j_1, \dots, j_{B_t})$ enumerate $\mathcal{I}_t$. With these definitions,

$$\mathbf{C}_t^{(B)} = \vec{K}_t\vec{K}_t^{\top}, \qquad \mathbf{N}_t^{(B)} = \vec{K}_t\vec{V}_t^{\top}, \qquad \bar{\mathbf{C}}_t^{(B)} = \frac{1}{B_t}\vec{K}_t\vec{K}_t^{\top}, \qquad \bar{\mathbf{N}}_t^{(B)} = \frac{1}{B_t}\vec{K}_t\vec{V}_t^{\top},$$

exactly for all t. (Equivalently, one may pad $\vec{K}_t, \vec{V}_t$ with zero columns up to width B; the products above are unchanged.) Substituting into Eq. (F.8) yields the canonical left-multiplicative low-rank form

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \underbrace{\Big(\frac{\eta_t}{B_t}\vec{K}_t\Big)}_{\vec{A}_t}\underbrace{(-\vec{K}_t^\top)}_{\vec{B}_t^\top}\mathbf{S}_{t-1} + \underbrace{\Big(\frac{\eta_t}{B_t}\vec{K}_t\Big)}_{\vec{A}_t}\underbrace{\vec{V}_t^\top}_{\widetilde{\vec{B}}_t^\top} \tag{F.11}$$

Thus, with ParallelFlow rank parameter $R = B_t \le B$ (or fixed rank $R = B$ after zero-padding), the drivers in Eq. (F.3) are

$$\vec{A}_t = \frac{\eta_t}{B_t}\vec{K}_t \in \mathbb{R}^{d\times B_t}, \qquad \vec{B}_t = -\vec{K}_t \in \mathbb{R}^{d\times B_t}, \qquad \widetilde{\vec{B}}_t = \vec{V}_t \in \mathbb{R}^{d_v\times B_t}.$$

In Algorithm 4, after the positive-decay renormalization of Eq. (4.15), one simply replaces η_t by $\widehat{\eta}_t$ and rescales the entire step-t value block by the common factor $(\delta_{i-1}^{(k)})^{-1}$ inside chunk k.

Complexity. Because FALCON-3 can be exactly cast as a CDE with rank-B drivers, we can utilize the `tensorInv` algorithm. For the small sliding windows used in our experiments (e.g., $B = 4$), the rank-B overhead is modest in practice and remains far cheaper than full $\mathcal{O}(L^2)$ attention while avoiding dense $d_x \times d_x$ state propagation. This derivation confirms that FALCON-3 benefits from the log-linear scaling of parallel scan methods while maintaining the expressive power of a sliding-window regressor.