Rethinking Expressivity and Efficiency in Test-Time Training

Zeyun Zhong[1,3] **Joya Chen**[2] **Manuel Martin**[3]
Frederik Diederichs[3] **Juergen Gall**[4,5] **Juergen Beyerer**[1,3]

[1]Karlsruhe Institute of Technology (KIT)
[2]National University of Singapore
[3]Fraunhofer IOSB
[4]Lamarr Institute for Machine Learning and Artificial Intelligence
[5]University of Bonn

Abstract

Test-Time Training (TTT) enables long-context processing via continuous weight updates during inference, but current methods struggle to balance the hardware efficiency of chunk-wise approximations with the expressivity of per-token update dynamics such as learning rate and momentum. We propose E^2-TTT (Expressive and Efficient TTT) to bridge this gap. Following prior chunk-wise TTT methods, we compute all gradients within a chunk at the chunk-start weights. Under this setting, we derive a closed-form chunk update that retains these per-token dynamics and exactly reproduces the chunk-end state of the corresponding token-wise recurrence. This enables fully parallelized chunk-level training while preserving the temporal structure of the update rule that prior chunk-wise methods discard. We validate E^2-TTT by training models up to 1.3B parameters from scratch. It performs on par with previous TTT and hybrid attention baselines in language modeling while outperforming them on in-context retrieval. Its advantage is most pronounced in length extrapolation: on the standard "Needle in a Haystack" passkey test, it retains over 90% accuracy at $8\times$ the training context length. Meanwhile, E^2-TTT can match the training throughput of efficient chunk-wise methods, demonstrating that it effectively reconciles expressivity with efficiency. The code is available at https://github.com/zeyun-zhong/E2-TTT.

1 Introduction

Many applications of large language models (LLMs), including long-horizon agentic tasks [16], require processing contexts that grow continually over time. Standard Transformers retain this history in a key–value (KV) cache [34] and repeatedly attend over it [56], causing memory and computation costs to grow with sequence length. External agent-memory systems mitigate this burden by organizing and selectively retrieving past experiences [60], but the retrieved memories must still be reintroduced into the model context and therefore do not change the underlying attention and KV-cache scaling. This motivates architectures that incorporate incoming information online into a compact recurrent state without explicitly retaining every preceding token [21, 31, 49, 52].

Compact-state models, including linear attention [31, 63], state-space models (SSMs) [13, 21], and recurrent components of hybrid architectures [32, 44], compress history into a fixed-size recurrent state for linear-time processing and parallel training. Gated delta-rule variants enable selective forgetting and targeted overwriting [62], but finite state capacity still causes memory collisions [10]. From a test-time regression perspective, Gated DeltaNet and test-time training (TTT) share an online-learning interpretation [57, 62]: the former updates a linear associative memory, whereas parametric

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

TTT stores context in the weights of a nonlinear inner learner, such as an MLP, and can use different online optimizers [52]. These *fast weights* [49] are updated by gradient steps on a self-supervised loss as the sequence is processed. Both approaches remain fixed-size states [52, 62], although deep nonlinear fast-weight memories have shown promise in language modeling and long-context retrieval [7, 8] and multimodal generation [12].

Along the update-granularity spectrum, token-wise TTT lies at the most expressive end [7, 52]: online gradient descent evaluates each token's gradient at fast weights shaped by all preceding updates, yielding a larger effective search space than batch gradient descent [52], while replacing token-dependent learning-rate, momentum, and decay factors with chunk-shared values loses expressive power [7]. This fine-grained state dependence, however, makes the inner loop sequential [52, 53], and small update batches lead to poor parallelism and low hardware utilization [66]. LaCT adopts the opposite operating point, computing the gradient of a summed loss over a large chunk and performing one fast-weight update per chunk, thereby substantially improving GPU utilization [66]. However, LaCT averages per-token momentum factors into a single chunk-level scalar, omitting the position-dependent gradient weighting that token-wise momentum induces within each chunk [66].

Motivated by this trade-off, we introduce E^2-TTT (Expressive and Efficient TTT), a chunk-parallel update that preserves per-token learning rate, momentum, and decay. Following prior mini-batch and chunk-wise TTT formulations [52, 66], we evaluate the gradients of all tokens in a chunk at the fast weights inherited from the previous chunk. Under this setting, we derive a closed-form chunk update with two scalar kernels that yields exactly the same fast weights and momentum at the end of each chunk as the corresponding token-wise recurrence. The kernels depend only on per-token scalars and are computed in parallel by log-space cumulative sums, eliminating the sequential fast-weight recurrence without materializing intermediate matrix states while retaining chunk-level hardware efficiency. We instantiate E^2-TTT in a hybrid architecture in which sliding-window attention captures fine-grained local interactions and the TTT state compresses history across chunks.

We empirically validate E^2-TTT by training models of up to 1.3B parameters from scratch on 15B tokens from the HuggingFace FineWeb-Edu dataset [43], comparing against strong linear attention and hybrid baselines. Our evaluation spans general language modeling, in-context retrieval, and length extrapolation, where the gains concentrate in the retrieval and extrapolation regimes. Notably, our model exhibits strong robustness in length extrapolation, maintaining $> 90\%$ accuracy on passkey retrieval in the standard "Needle in a Haystack" test at $8\times$ the training context, a regime where prior chunk-based methods collapse. Finally, we explore the versatility of our approach through parameter-efficient adaptation for long-context video understanding, achieving performance comparable to full fine-tuning at the 2B-parameter scale on standard multimodal benchmarks.

2 Background: Test-Time Training

Conventional recurrent layers compress context into fixed-size vector- or matrix-valued activations. Parametric Test-Time Training (TTT) instead uses the parameters $\mathbf{W}_t$ of an inner learner $f_{\mathbf{W}_t}(\cdot) : \mathbb{R}^d \to \mathbb{R}^d$ as the recurrent state [52]. These *fast weights* [49] are updated by a self-supervised inner loop on each sequence during both training and inference, whereas the *slow weights* are learned by the outer loop and remain fixed at inference. For notational simplicity, we write $\mathbf{W}_t \in \mathbb{R}^{d\times d}$ as a single matrix. For multi-layer learners, $\mathbf{W}_t$ denotes the full collection of fast-weight tensors.

As illustrated in Figure 1, the general TTT process operates in two stages per step: an update stage and an output stage. First, input tokens are projected into query ($\boldsymbol{q}_t$), key ($\boldsymbol{k}_t$), and value ($\boldsymbol{v}_t$) vectors, where $\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t \in \mathbb{R}^d$. The keys and values are used to update the fast weights via a self-supervised objective, effectively encoding the context into a neural memory with fixed state size. Second, the updated fast weights are applied to the query to generate the output token $\boldsymbol{o}_t$. Based on how these update and output rules are structured, we categorize current TTT formulations into three groups (see Table 1).

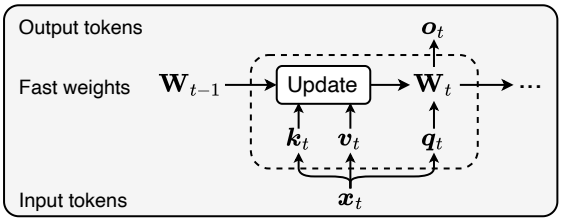

Figure 1: Token-wise TTT. At each step, the pair $(\boldsymbol{k}_t, \boldsymbol{v}_t)$ defines a self-supervised loss whose gradient step (the "training" stage) updates $\mathbf{W}_{t-1}$ to $\mathbf{W}_t$. The query $\boldsymbol{q}_t$ then reads from $\mathbf{W}_t$ to produce $\boldsymbol{o}_t$.

Table 1: Comparison of TTT update schedules. *Gradient evaluated at*: the fast-weight state used to compute the per-token gradients $\mathbf{G}_t$. *State recurrence*: the granularity at which the states $(\mathbf{W}_t, \mathbf{M}_t)$ evolve. *Output computed from*: the state applied to the query $\boldsymbol{q}_t$. E^2-TTT matches Chunk-TTT on both efficiency-relevant columns while retaining a token-wise state recurrence.

Method	Gradient Evaluated at	State Recurrence	Output Computed from	Efficient Training
Token-wise TTT [52]	Previous token	Token-wise	Current token	✗
Mini-Batch TTT [52, 7]	Batch start	Token-wise	Current token	✗
Chunk-TTT (LaCT) [66]	Chunk start	Chunk-wise	Chunk start	✓
E^2-TTT (Ours)	Chunk start	Token-wise	Chunk start	✓

Expressive Token-wise Update. In this standard formulation, both the gradient evaluation and the state update are token-wise. At each time step t, the fast weights $\mathbf{W}_t$ are updated by minimizing a self-supervised reconstruction loss $\mathcal{L}$ between the transformed key $f_{\mathbf{W}_{t-1}}(\boldsymbol{k}_t)$ and the value $\boldsymbol{v}_t$. Let $\mathbf{G}_t$ denote the negative gradient, and η_t the learning rate:

$$\mathbf{G}_t = -\nabla_{\mathbf{W}} \mathcal{L}\big(f_{\mathbf{W}_{t-1}}(\boldsymbol{k}_t), \boldsymbol{v}_t\big), \qquad \mathbf{W}_t = \mathbf{W}_{t-1} + \eta_t \mathbf{G}_t. \tag{1}$$

The loss function is commonly Mean Squared Error [52]. The updated fast weights $\mathbf{W}_t$ are then immediately used to compute the output vector $\boldsymbol{o}_t$ for the current query $\boldsymbol{q}_t$:

$$\boldsymbol{o}_t = f_{\mathbf{W}_t}(\boldsymbol{q}_t). \tag{2}$$

Token-wise Update with Mini-Batch Gradient. While the token-wise update offers high expressivity [52], it cannot parallelize gradient evaluation, since $\mathbf{G}_t$ depends on $\mathbf{W}_{t-1}$. Consequently, recent work [52, 7] adopts a mini-batch approach. The sequence is partitioned into mini-batches indexed by r, each of small size B (e.g., 16). We use the notation $(\cdot)_t^{[r]}$ to denote a variable associated with the t-th token within the r-th batch. Let $\mathbf{W}_B^{[r-1]} = \mathbf{W}_0^{[r]}$ denote the batch-start state inherited from the previous batch. Inside batch r, gradients are approximated using this fixed $\mathbf{W}_B^{[r-1]}$, which decouples forward pass dependencies:

$$\mathbf{G}_t^{[r]} = -\nabla_{\mathbf{W}} \mathcal{L}\big(f_{\mathbf{W}_B^{[r-1]}}(\boldsymbol{k}_t^{[r]}), \boldsymbol{v}_t^{[r]}\big), \quad \mathbf{W}_t^{[r]} = \mathbf{W}_{t-1}^{[r]} + \eta_t \mathbf{G}_t^{[r]}, \quad \boldsymbol{o}_t^{[r]} = f_{\mathbf{W}_t^{[r]}}(\boldsymbol{q}_t^{[r]}) \tag{3}$$

This approximation enables the parallel computation of gradients $\mathbf{G}_t^{[r]}$ for all tokens $t \in \{1, \dots, B\}$. Note that only the state used for gradient evaluation is shared, while the resulting fast weights $\mathbf{W}_t^{[r]}$ and outputs $\boldsymbol{o}_t^{[r]}$ remain token-indexed. Once calculated, the fast weights $\mathbf{W}_t^{[r]}$ can be computed, e.g., via a parallel associative scan [50].

Efficient Chunk-wise Update. Despite enabling parallel gradient calculation, the small mini-batch size and the token-wise dependency in mini-batch update still result in low hardware utilization (often $< 5\%$ FLOPs efficiency [66]). LaCT [66] therefore partitions the sequence into large chunks indexed by r, each of size $C \gg B$ (e.g., $C = 512$). Within the r-th chunk, LaCT evaluates all token losses at the chunk-start weights $\mathbf{W}^{[r-1]}$ and leverages the linearity of differentiation to aggregate the optimization objective. Instead of materializing gradients at every token step, it computes the gradient of the accumulated weighted loss, producing a single update direction $\mathbf{G}^{[r]}$ for the entire chunk. When momentum is integrated, LaCT adopts a simplified strategy: averaging the time-dependent momentum factors $\beta_t^{[r]}$ to maintain a chunk-level momentum buffer $\mathbf{M}^{[r]}$:

$$\mathbf{G}^{[r]} = -\nabla_{\mathbf{W}} \left(\sum_{t=1}^{C} \eta_t^{[r]} \mathcal{L}\big(f_{\mathbf{W}^{[r-1]}}(\boldsymbol{k}_t^{[r]}), \boldsymbol{v}_t^{[r]}\big) \right), \tag{4}$$

$$\mathbf{M}^{[r]} = \left(\frac{1}{C} \sum_{t=1}^{C} \beta_t^{[r]} \right) \mathbf{M}^{[r-1]} + \mathbf{G}^{[r]}, \qquad \mathbf{W}^{[r]} = \mathbf{W}^{[r-1]} + \mathbf{M}^{[r]}. \tag{5}$$

This formulation allows maintaining chunk-level states (i.e., $\mathbf{G}^{[r]}, \mathbf{M}^{[r]}, \mathbf{W}^{[r]}$) instead of token-level states, reducing the number of materialized fast weight states from C to 1 per chunk. As the new weights $\mathbf{W}^{[r]}$ are only available after processing the entire chunk, the output $\mathbf{O}^{[r]}$ (comprising outputs for all tokens in the chunk) is computed using the weights from the previous chunk. This leaves the fast-weight path blind to intra-chunk history, but renders the chunk forward pass fully parallel:

$$\mathbf{O}^{[r]} = f_{\mathbf{W}^{[r-1]}}(\mathbf{Q}^{[r]}). \tag{6}$$

3 Expressive and Efficient Test-Time Training

Token-wise TTT retains fine-grained optimizer dynamics but is inherently sequential, whereas chunk-wise TTT gains efficiency by collapsing these dynamics into a single update, obscuring the temporal variance of token importance within a chunk. We introduce E^2-TTT (Expressive and Efficient TTT), a closed-form chunk update that preserves the effects of per-token learning rate, momentum, and decay under shared chunk-start gradient evaluation. It exactly recovers the corresponding chunk-end fast-weight and momentum states while retaining large-chunk parallelism (see Table 1). For simplicity, we omit the current-chunk superscript $[r]$ in this section (*e.g.* $\mathbf{W}_t = \mathbf{W}_t^{[r]}$). We index the updates within a chunk by $t \in \{1, \dots, C\}$, with $t = 0$ denoting the start state inherited from the previous chunk ($\mathbf{W}_0 = \mathbf{W}_C^{[r-1]}$, $\mathbf{M}_0 = \mathbf{M}_C^{[r-1]}$).

3.1 Primal Formulation: Token-wise Dynamics

We first define the token-wise state recurrence that serves as our reference. Unlike the simplified chunk update in Eq. 5, it retains per-token momentum and multiplicative decay, motivated by recent work [7, 62]. Let $\mathbf{W}_t, \mathbf{M}_t, \mathbf{G}_t \in \mathbb{R}^{d\times d}$ denote the fast weights, momentum, and negative gradient (update direction), respectively. Let $\gamma_t, \beta_t \in (0, 1)$ be the scalar decay and momentum factors, and $\eta_t > 0$ be the learning rate. Our targeted sequential recurrence is defined as:

$$\mathbf{M}_t = \beta_t \mathbf{M}_{t-1} + \eta_t \mathbf{G}_t, \qquad \mathbf{W}_t = \gamma_t \mathbf{W}_{t-1} + \mathbf{M}_t. \tag{7}$$

The specific parameterizations of γ_t, β_t, and η_t are predicted from the input $\boldsymbol{x}_t$ independently of $(\mathbf{W}, \mathbf{M})$, as described in Section 4. As in the mini-batch setting (Eq. 3), the gradients $\mathbf{G}_t$ are computed using the fixed weights from the previous chunk ($\mathbf{W}_C^{[r-1]}$), which decouples them across t and allows them to be computed in parallel before the update step. Only the chunk-end states $(\mathbf{W}_C^{[r]}, \mathbf{M}_C^{[r]})$ are propagated to the next chunk. The intermediate states $(\mathbf{W}_t^{[r]}, \mathbf{M}_t^{[r]})$ for $t < C$ are not stored.

3.2 Closed-Form Parallelization via Scalar Kernels

A naive execution of Eq. (7) requires sequential iteration ($t = 1 \to C$). While parallel prefix scans could theoretically compute the states, they typically necessitate computing and storing $\mathbf{W}_t$, $\mathbf{M}_t$, and $\mathbf{G}_t$ at every time step. Since these are dense matrices, storing the full sequence incurs a prohibitive memory cost of $O(C \cdot d^2)$, causing bottlenecks on modern accelerators, especially given the large chunk size C. To enable efficient training without simplifying the dynamics of Eq. (7), we derive a closed-form equivalent that "jumps" from the chunk start ($t = 0$) to the end ($t = C$) in a single step. Our insight is that while the matrix states $(\mathbf{W}, \mathbf{M})$ are expensive, the decay and momentum factors (γ_t, β_t) are scalars. By unrolling the recurrence and re-indexing the summation, we can compress the entire history of temporal dynamics into scalar coefficients that isolate the contribution of each gradient $\mathbf{G}_t$ to the final state (see Appendix A for the full derivation):

$$\mathbf{W}_C = \underbrace{\Big(\prod_{t=1}^{C} \gamma_t\Big)}_{\text{Decay term}} \mathbf{W}_0 + \underbrace{\sum_{t=1}^{C} \Big(\Big(\prod_{i=1}^{t} \beta_i\Big)\mathbf{M}_0 \prod_{q=t+1}^{C} \gamma_q\Big)}_{\text{Momentum term}} + \underbrace{\sum_{t=1}^{C} \eta_t \mathbf{G}_t \sum_{j=t}^{C} \Big(\prod_{i=t+1}^{j} \beta_i \prod_{q=j+1}^{C} \gamma_q\Big)}_{\text{Gradient term}}. \tag{8}$$

The inner summation in the gradient term encapsulates the cumulative impact of the t-th token on the final chunk weight. We formalize this by defining efficient suffix products $\widetilde{\beta}_t, \widetilde{\gamma}_t$ and a ratio sum R_t:

$$\widetilde{\beta}_t := \prod_{i=t+1}^{C} \beta_i, \quad \widetilde{\gamma}_t := \prod_{i=t+1}^{C} \gamma_i, \quad R_t := \sum_{i=t}^{C} \frac{\widetilde{\gamma}_i}{\widetilde{\beta}_i}, \tag{9}$$

with the convention $\prod_{i=C+1}^{C}(\cdot) = 1$. These terms depend only on the scalars β_t, γ_t and can therefore be computed efficiently in parallel (e.g., via log-space cumulative sums) with negligible cost. Rewriting the prefix products in Eq. (8) via the identity $\prod_{i=1}^{j} \beta_i = \big(\prod_{i=1}^{C} \beta_i\big)/\widetilde{\beta}_j$ and unrolling the momentum recurrence in parallel yields the following result.

Algorithm 1 E^2-TTT chunk update.

Require: chunk-start state $(\mathbf{W}_0, \mathbf{M}_0)$; tokens $\{(\boldsymbol{q}_t, \boldsymbol{k}_t, \boldsymbol{v}_t)\}_{t=1}^{C}$; scalars $\{\eta_t, \beta_t, \gamma_t\}_{t=1}^{C}$

1: $\boldsymbol{o}_t = f_{\mathbf{W}_0}(\boldsymbol{q}_t), \quad \mathcal{L}_t = \mathcal{L}\big(f_{\mathbf{W}_0}(\boldsymbol{k}_t), \boldsymbol{v}_t\big)$ // Compute outputs and per-token losses

2: Compute scalar aggregates $\widetilde{\beta}_t, \widetilde{\gamma}_t, R_t$ via log-space cumulative sums; form $\mathcal{K}_t^W, \mathcal{K}_t^M$

3: $\boldsymbol{g}_t \leftarrow -\partial \mathcal{L}_t / \partial f_{\mathbf{W}_0}(\boldsymbol{k}_t)$ // Per-token activation gradients (single backward)

4: $\Delta_W \leftarrow \textsc{Aggregate}\big(\{\mathcal{K}_t^W\}, \{\boldsymbol{g}_t\}, \{\boldsymbol{k}_t\}\big)$ // $\equiv -\sum_t \mathcal{K}_t^W \nabla_{\mathbf{W}} \mathcal{L}_t\big|_{\mathbf{W}_0}$

5: $\Delta_M \leftarrow \textsc{Aggregate}\big(\{\mathcal{K}_t^M\}, \{\boldsymbol{g}_t\}, \{\boldsymbol{k}_t\}\big)$ // $\equiv -\sum_t \mathcal{K}_t^M \nabla_{\mathbf{W}} \mathcal{L}_t\big|_{\mathbf{W}_0}$; reuses $\{\boldsymbol{g}_t\}$ from line 3

6: $\mathbf{W}_C \leftarrow \widetilde{\gamma}_0 \mathbf{W}_0 + \widetilde{\beta}_0 R_1 \mathbf{M}_0 + \Delta_W$

7: $\mathbf{M}_C \leftarrow \widetilde{\beta}_0 \mathbf{M}_0 + \Delta_M$

8: **return** outputs $\{\boldsymbol{o}_t\}$, next-chunk state $(\mathbf{W}_C, \mathbf{M}_C)$

Proposition 3.1 (Exact chunk-end states)**.** *Under the recurrence of Eq. (7) with $\mathbf{G}_t$ evaluated at the frozen chunk-start $\mathbf{W}_0$ and scalars $\eta_t, \beta_t, \gamma_t$ independent of $(\mathbf{W}, \mathbf{M})$, the chunk-end states satisfy*

$$\mathbf{M}_C = \widetilde{\beta}_0\, \mathbf{M}_0 + \sum_{t=1}^{C} \mathcal{K}_t^M\, \mathbf{G}_t, \qquad \mathcal{K}_t^M := \eta_t\, \widetilde{\beta}_t, \tag{10}$$

$$\mathbf{W}_C = \widetilde{\gamma}_0\, \mathbf{W}_0 + \widetilde{\beta}_0 R_1\, \mathbf{M}_0 + \sum_{t=1}^{C} \mathcal{K}_t^W\, \mathbf{G}_t, \qquad \mathcal{K}_t^W := \eta_t\, \widetilde{\beta}_t\, R_t. \tag{11}$$

Two scalar kernels with distinct temporal weighting. Proposition 3.1 shows that the chunk-end weights $\mathbf{W}_C$ and the chunk-end momentum $\mathbf{M}_C$ aggregate the per-token gradients $\{\mathbf{G}_t\}$ through two distinct scalar kernels. The momentum kernel $\mathcal{K}_t^M = \eta_t \widetilde{\beta}_t$ measures how much of $\mathbf{G}_t$ survives to the chunk end through the momentum decay $\widetilde{\beta}_t$. The weight kernel $\mathcal{K}_t^W = \eta_t \widetilde{\beta}_t R_t$ additionally weights this by the cumulative weight-decay survival R_t, capturing how the gradient propagates through both the momentum buffer and the subsequent weight updates. A bound analysis of R_t and the resulting kernels $\mathcal{K}_t^W, \mathcal{K}_t^M$ under near-unit β_t, γ_t is provided in Appendix A.3. In contrast, LaCT [66] carries only η_t in the gradient term (without $\widetilde{\beta}_t$ or R_t) and collapses the momentum factor to a chunk average, removing the position-dependent weighting within a chunk (Eq. 5).

One backward pass for both aggregates. Computing the two aggregates $\sum_t \mathcal{K}_t^W \mathbf{G}_t$ and $\sum_t \mathcal{K}_t^M \mathbf{G}_t$ would naively require two backward passes. We collapse them to one by noting that the kernels differ only in scalar coefficients and share the per-token activation gradients $\boldsymbol{g}_t = -\partial \mathcal{L}_t / \partial f_{\mathbf{W}_0}(\boldsymbol{k}_t)$ (see Appendix A.4). These gradients arise as a $C \times d$ intermediate of LaCT's backward pass, which contracts them into a $d \times d$ weight gradient and then discards them. We instead keep this intermediate and use it for two contractions. The marginal cost over LaCT is therefore a single additional re-weighted contraction. Algorithm 1 summarizes the procedure.

The output is computed using the chunk-start weights, $\mathbf{O}^{[r]} = f_{\mathbf{W}_0}(\mathbf{Q}^{[r]})$, consistent with chunk-wise TTT and ensuring a fully parallel forward pass. Algorithm 1 retains LaCT's $O(C \cdot d^2)$ per-chunk time complexity: both weighted gradient contractions are $O(C \cdot d^2)$, while the scalar scans add only $O(C)$. We report measured throughput in Appendix B.5. Cross-chunk equivalence to the original sequential recurrence follows by induction, since $(\mathbf{W}_C, \mathbf{M}_C)$ is precisely the state propagated to the next chunk. We numerically verify this on a 128-chunk trajectory ($C = 512$, 65,536 tokens), where the relative L_2 deviation from the fully sequential reference is below 2×10^{-6} on both $\mathbf{W}_C$ and $\mathbf{M}_C$, within FP32 round-off accumulated over the trajectory.

4 E^2-TTT Model Architecture

The chunk-wise output rule in Eq. (6) applies one fast-weight state to every token in a chunk. Consequently, a token cannot influence later tokens in the same chunk through the TTT state. We compensate for this missing within-chunk path with an established hybrid design [3, 26, 41, 66]: E^2-TTT carries information across chunks, while Sliding Window Attention (SWA) models local causal interactions (Fig. 2).

Overall Architecture. The macro-architecture of E^2-TTT follows a Llama-style backbone [54], interleaving token-mixing layers with SwiGLU feed-forward blocks. Given $\mathbf{X} \in \mathbb{R}^{L\times d}$, each token-mixing layer projects shared queries $\mathbf{Q}$, keys $\mathbf{K}$, and values $\mathbf{V}$ and sends them to two parallel branches: SWA for local retrieval and E^2-TTT for cross-chunk context modeling. For the TTT branch, we follow prior hybrid and chunk-wise TTT models [28, 66], applying SiLU to the queries and keys and then L2-normalizing them for stability. Lightweight linear heads predict the token-dependent learning rate η_t, momentum factor β_t, and decay strength α_t from $\mathbf{X}$. The decay factor γ_t is then derived from η_t and α_t, as defined below. Following HQLT [28], a token-dependent gate projected from $\mathbf{X}$ interpolates the two branch outputs element-wise. The fused representation is normalized and passed through the output projection.

Input-Dependent Dynamics. We parameterize $\eta_t, \beta_t, \gamma_t$ as functions of $\boldsymbol{x}_t \in \mathbb{R}^d$, allowing the update dynamics to vary across tokens. The corresponding projections are learned end-to-end from the language-modeling objective, without auxiliary supervision. Appendix B.1 gives the base scalars, Appendix B.3 evaluates their sensitivity, and Appendix B.4 analyzes the learned coefficients.

The learning rate η_t modulates the update magnitude. Following TTT [52], we bound it by a base rate η_{base}:

$$\eta_t = \eta_{\text{base}} \cdot \sigma(\text{Linear}_\eta(\boldsymbol{x}_t)). \tag{12}$$

The momentum factor β_t controls how strongly past gradients persist. To encourage long-term memory retention, a timescale τ biases it toward values near 1, following gated recurrent parameterizations [61, 51]:

$$\beta_t = \sigma(\text{Linear}_\beta(\boldsymbol{x}_t))^{\frac{1}{\tau}}. \tag{13}$$

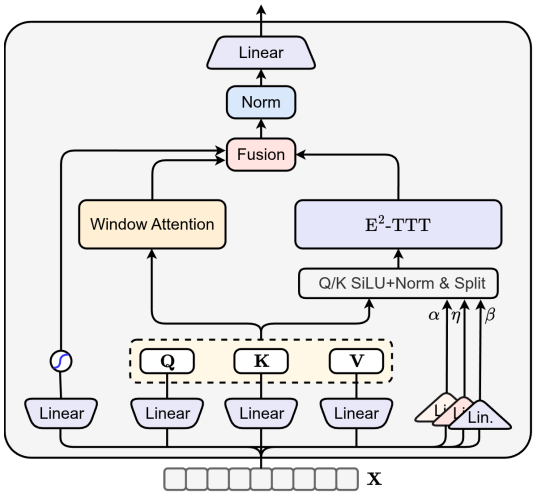

Figure 2: E^2-TTT token mixing block. SWA and E^2-TTT run in parallel over shared $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ and are fused by a data-dependent gate. The TTT update is modulated by input-dependent α, η, β.

The decay factor γ_t serves as a forget gate. To prevent excessive regularization when the update step is small, we predict a bounded decay strength α_t and couple it to the learning rate η_t, yielding the standard multiplicative form of decoupled weight decay [39]:

$$\alpha_t = \alpha_{\text{base}} \cdot \sigma(\text{Linear}_\alpha(\boldsymbol{x}_t)), \quad \gamma_t = 1 - \eta_t \alpha_t. \tag{14}$$

Fast-weight Network. The formulation supports a general differentiable fast-weight learner f. We instantiate it with the two MLPs used in our experiments [52, 66]: a GELU MLP and a SwiGLU MLP, both with a residual connection and LayerNorm (LN). The GELU variant uses two fast-weight matrices:

$$f_{\mathbf{W}}(\boldsymbol{x}) = \boldsymbol{x} + \text{LN}\big(\mathbf{W}_2\, \text{GELU}(\mathbf{W}_1 \boldsymbol{x})\big). \tag{15}$$

The SwiGLU variant uses three fast-weight matrices:

$$f_{\mathbf{W}}(\boldsymbol{x}) = \boldsymbol{x} + \text{LN}\big(\mathbf{W}_3\, (\text{SiLU}(\mathbf{W}_1 \boldsymbol{x}) \odot (\mathbf{W}_2 \boldsymbol{x}))\big). \tag{16}$$

5 Experiments

We evaluate our method across three distinct categories of tasks to verify its versatility and robustness: general language modeling, to validate overall capabilities on standard benchmarks (Sec. 5.1); in-context retrieval, to assess the model's precise recall abilities (Sec. 5.2); and length extrapolation, to test generalization to sequence lengths exceeding the training horizon (Sec. 5.3). We then isolate the contribution of the update rule itself in a tightly matched ablation (Sec. 5.4). Finally, we extend our evaluation to long-context video understanding, exploring the potential of our E^2-TTT to leverage long-range temporal dependencies in multimodal data (Sec. 5.5).

Setup. Following HQLT [28], we train language models with either 340M or 1.3B trainable parameters (using sequence lengths of 2048 and 2240, respectively) from scratch on 15B tokens of the HuggingFace FineWeb-Edu dataset [43]. These choices mostly follow the configurations of the recent work on linear transformers [61, 63, 20], including the choice of tokenizer

Table 2: General language modeling experiments. We report perplexity (↓) and zero-shot accuracy (↑) for models with 340M and 1.3B parameters, all trained on 15B tokens. E^2-TTT attains the lowest perplexity at both scales, while all methods lie within about two points of one another in average zero-shot accuracy. Results marked with * are cited from HQLT.

Model	Attn. window	Wiki. ppl ↓	LMB. ppl ↓	LMB. acc ↑	PIQA acc ↑	Hella. acc_n ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	Avg. ↑
340M params										
Transformer++*	2048	26.5	34.9	33.9	67.6	41.0	53.7	60.2	29.0	47.6
DeltaNet*	–	27.6	35.0	32.8	67.1	40.8	52.6	58.5	28.8	46.8
HQLT*	512	27.0	28.0	35.9	66.3	41.3	53.2	60.1	29.0	47.6
LaCT	512	27.0	28.2	**37.1**	66.8	41.6	51.4	59.0	30.0	47.7
E^2-TTT$_{\text{SwiGLU}}$	512	26.4	27.8	36.6	**67.7**	42.2	51.6	61.4	28.3	48.0
E^2-TTT$_{\text{MLP}}$	512	**25.5**	**25.0**	37.0	67.4	**43.9**	**54.9**	**61.3**	**30.2**	**49.1**
1.3B params										
Transformer++*	2048	19.8	17.9	42.6	71.0	50.3	55.8	65.2	33.2	53.0
DeltaNet*	–	20.6	19.9	39.3	70.1	49.5	52.5	68.5	34.2	52.3
Mamba2	–	20.5	17.0	41.1	71.0	51.5	54.5	66.5	35.0	53.3
HQLT	512	20.3	16.1	42.6	71.0	51.1	55.1	**69.2**	34.0	53.8
LaCT	512	20.1	16.7	42.0	70.4	51.6	55.4	65.6	33.1	53.0
E^2-TTT$_{\text{SwiGLU}}$	512	19.9	15.8	43.2	71.1	51.9	54.2	67.5	34.1	53.6
E^2-TTT$_{\text{MLP}}$	512	**19.5**	**15.3**	**43.7**	**71.4**	**52.7**	**56.1**	67.6	**35.2**	**54.5**

(`fla-hub/transformer-1.3B-100B`). The subword-unit vocabulary size is 32K for all models. All models have 24 layers. The baseline Transformer architecture is from Touvron et al. [54] (denoted as Transformer++, following prior convention) and the DeltaNet configuration is from Yang et al. [63]. We also compare against two hybrid models: HQLT and LaCT [66], both of which utilize window attention combined with DeltaNet and chunk-wise TTT, respectively. For all baseline models, we either report the results from the respective papers or train them using official public repositories under identical conditions to E^2-TTT for fair comparison. We use a chunk size of 512 for both LaCT and our models. Full architectural, training, and evaluation details are in Appendix B.1–B.2. Appendix B.7 additionally reports a feasibility run with a larger training budget.

5.1 General Language Modeling Capabilities

In Table 2, we present the language modeling perplexity and zero-shot accuracy on commonsense reasoning benchmarks for models with 340M and 1.3B parameters. E^2-TTT obtains the lowest perplexity at both scales, with the margin most visible on LAMBADA (15.3 vs. 16.1 for the strongest baseline at 1.3B, and 25.0 vs. 28.0 at 340M). Gains in zero-shot accuracy are smaller: E^2-TTT$_{\text{MLP}}$ reaches 54.5% average at 1.3B against 53.8% for the strongest baseline, and leads on five of six tasks while trailing HQLT and DeltaNet on ARC-e.

5.2 Evaluating In-Context Retrieval Abilities

In this section, we evaluate the performance of E^2-TTT on recall-intensive tasks. Following prior work [63, 2, 28], we focus on three tasks: FDA [2], SWDE [38], and SQuAD [46]. Table 3 summarizes the results for the 340M and 1.3B parameter scales. Consistent with Irie et al. [28], full attention models (Transformer++) still hold an advantage (53.3% average) due to their explicit history access. However, E^2-TTT substantially narrows this gap: our 1.3B SwiGLU model (43.6%) surpasses HQLT (35.5%) and LaCT (36.7%) on average, demonstrating the benefit of our expressive chunk-wise update rule. Our SwiGLU variant also outperforms the MLP variant by a large margin (43.6% vs. 37.1% average), indicating that gating mechanisms are beneficial for precise information extraction.

Table 3: Accuracy on real-world retrieval tasks with input truncated to 2K tokens. *: from HQLT.

Model	Attn. window	SWDE acc ↑	SQuAD acc ↑	FDA acc ↑	Avg.
340M params					
Transformer++*	2048	**44.9**	36.9	**52.3**	**44.7**
Transformer++*	1024	30.4	25.5	31.2	29.0
DeltaNet*	–	18.5	25.2	8.6	17.4
HQLT*	512	22.9	35.7	17.3	25.3
LaCT	512	25.0	39.0	22.2	28.7
E^2-TTT$_{\text{SwiGLU}}$	512	<u>35.4</u>	<u>39.6</u>	<u>26.1</u>	<u>33.7</u>
E^2-TTT$_{\text{MLP}}$	512	23.6	**40.1**	26.0	29.9
1.3B params					
Transformer++*	2048	**53.7**	41.5	**64.7**	**53.3**
DeltaNet*	–	32.9	29.9	23.6	28.8
Mamba2	–	33.8	37.3	26.1	32.4
HQLT	512	32.1	43.4	31.0	35.5
LaCT	512	39.2	<u>44.9</u>	26.0	36.7
E^2-TTT$_{\text{SwiGLU}}$	512	<u>47.3</u>	44.4	<u>39.0</u>	<u>43.6</u>
E^2-TTT$_{\text{MLP}}$	512	37.2	**45.6**	28.5	37.1

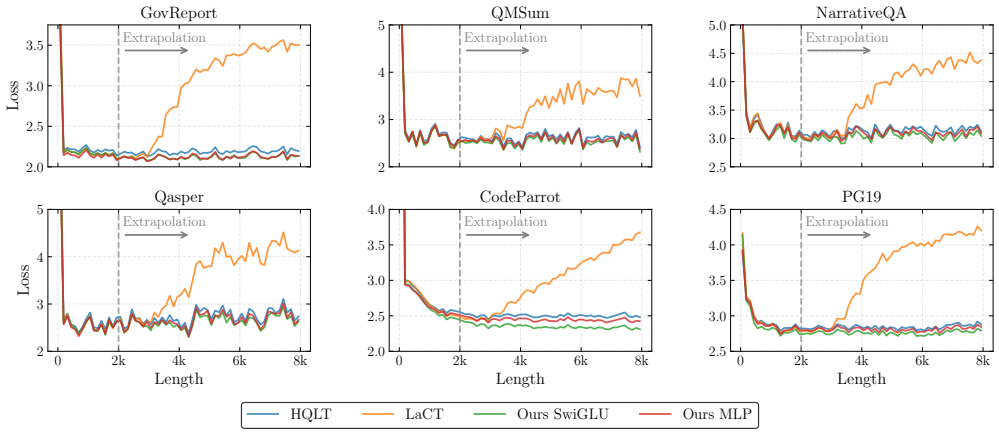

Figure 3: Length extrapolation of 1.3B models. We visualize the per-token loss metric [59] on six long-context benchmarks, where E^2-TTT maintains stability while the LaCT baseline diverges.

5.3 Evaluating Length Extrapolation Capabilities

A critical advantage of our E^2-TTT is its potential to generalize to sequence lengths significantly exceeding the training horizon. We evaluate this capability across three dimensions: language modeling stability, synthetic long-context retrieval, and real-world long-context benchmarks.

Language Modeling Extrapolation. We first assess whether the models can maintain stable language modeling performance when processing sequences longer than their training context (2K tokens). Figure 3 visualizes the per-token loss metric [59] on six long-context datasets. As shown in Fig. 3, LaCT struggles with length extrapolation, with loss exploding immediately past the training boundary on all six datasets. HQLT maintains stability but shows higher validation loss (e.g., on CodeParrot). In contrast, both E^2-TTT variants demonstrate lower and stable loss curves throughout the extrapolated region. This confirms that our expressive chunk-wise updates robustly generalize to unseen lengths.

Long-range Associative Recall. Stability alone does not guarantee information retention. To test effective capacity, we employ the "Single-Needle in a Haystack" (S-NIAH) benchmark [24], extending the test sequence up to 16K tokens (8× the training horizon). Figure 4 reveals a stark contrast in extrapolation behavior. On S-NIAH-1, both E^2-TTT variants retain at least 85% accuracy at 16K, while LaCT collapses to near-zero and HQLT degrades to 25%. On S-NIAH-2, E^2-TTT$_{\text{SwiGLU}}$ retains a clear lead through 16K, achieving 40.6% at 8K versus 15.4% (HQLT) and 5.6% (LaCT). The SwiGLU variant consistently outperforms the MLP variant in the extrapolation regime, indicating the benefit of a gated fast-weight network for handling complex retrieval patterns. A more detailed numerical comparison, including additional baselines, is provided in Appendix B.6.

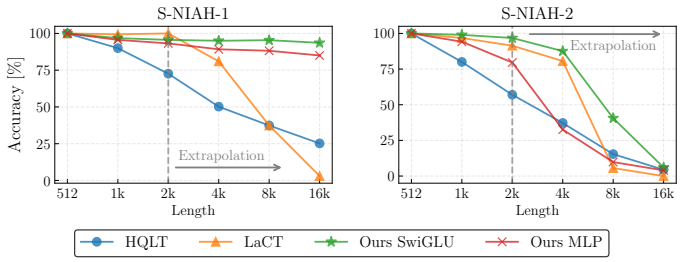

Figure 4: Zero-shot accuracy on S-NIAH-1 and S-NIAH-2 for 1.3B models. All methods use sliding window attention with a 512-token window. E^2-TTT maintains robust retrieval well beyond the training context, while LaCT collapses past 4K and HQLT degrades steadily across both tasks.

Real-world Long Context Understanding. Finally, we evaluate performance on LongBench [6], a comprehensive suite of 14 real-world long-context tasks. As detailed in Table 4, our method translates its length extrapolation advantages into practical gains. E^2-TTT$_{\text{SwiGLU}}$ achieves the highest average score (14.1%), outperforming HQLT (12.1%) and the SSM-based Mamba2 (10.3%), while nearly doubling the performance of LaCT (7.7%). The improvements are systematic across task categories: E^2-TTT$_{\text{SwiGLU}}$ outperforms Mamba2 on 13 of 14 tasks, HQLT on 11 of 14, and LaCT on all 14. The MLP variant similarly outperforms all baselines (13.3% average), confirming that the gains stem

Table 4: Evaluation on Longbench. Accuracy on 14 tasks from LongBench: Narrative QA, QasperQA, MultiField QA, HotpotQA, 2WikiMulti QA, Musique, GovReport, QMSum, MultiNews, TRec, Trivia QA, SamSum, LCC, and RepoBench-P by order.

Model (1.3B)	Single-Doc QA			Multi-Doc QA			Summarization			Few-shot			Code		Avg.
	NQA	QQA	MFQ	HQA	2WM	Mus	GvR	QMS	MNs	TRC	TQA	SSM	LCC	RBP	
Mamba2	2.0	4.6	11.4	3.5	8.9	2.8	4.7	16.6	7.7	9.5	16.6	13.0	22.3	20.0	10.3
HQLT	0.8	1.0	9.3	3.3	7.9	2.0	6.7	14.5	12.4	25.8	19.9	**23.1**	19.8	22.8	12.1
LaCT	1.1	4.5	8.2	3.5	4.7	2.4	9.4	5.1	11.5	10.0	5.6	7.6	19.4	14.5	7.7
E^2-TTT$_{SwiGLU}$	**2.8**	**4.9**	**14.2**	**5.0**	9.0	**3.8**	**10.3**	**16.7**	12.1	**34.0**	16.2	20.1	**24.4**	23.2	**14.1**
E^2-TTT$_{MLP}$	1.7	4.4	12.4	4.4	**10.4**	3.1	8.8	16.6	**13.6**	26.5	**20.3**	19.3	19.5	**24.6**	13.3

from our update rule rather than a specific gating choice. This consistency across single-document QA, multi-document QA, summarization, few-shot, and code tasks demonstrates that our expressive chunk-wise update rule enables E^2-TTT to leverage extended context more effectively than baselines spanning multiple sub-quadratic paradigms (SSM, linear attention, and TTT).

5.4 Ablation Study

To isolate the update rule from the surrounding architecture, we train two additional 1.3B models inside our own framework. Apart from the published LaCT configuration and the SWA-only reference, all rows of Table 5 share the same architecture, including SWA and the fusion gate, and differ only in how the inner loop is aggregated. LaCT-Matched ports LaCT's update rule into our framework. Chunk-averaged is our own model with the per-token momentum and decay factors collapsed to chunk-level scalars. LaCT-Matched improves over its default configuration on S-NIAH-1 within and near the training context (81.0→99.2 at 4K) and is slightly ahead of our own model at 2K, while on S-NIAH-2 it is mixed (98.6 vs. 91.4 at 2K, 65.8 vs. 80.6 at 4K). Beyond the training context, however, it still collapses to 0.0 at 16K. Chunk-averaged isolates the effect of the per-token scalars. Collapsing them drops S-NIAH-1 at 8K from 95.4 to 6.8, the level of the SWA baseline alone, and the degradation already begins inside the training context (95.6→25.2 at 2K). The collapsed model nonetheless trains normally and is even marginally ahead in-domain (15.6 vs. 15.8 LAMBADA perplexity, 53.8% vs. 53.6% zero-shot average). We therefore attribute the long-context retrieval gains, both within and beyond the training context, to our expressive chunk-wise update rule. Ablations of the remaining components, including inner-loop optimizer terms, chunk granularity, and sensitivity to the base scalars, are reported in Appendix B.3.

Table 5: Controlled comparison of update rules (1.3B). The last three rows share our framework and differ only in how the inner loop is aggregated. Training context is 2K tokens. Columns to the right of 2K are extrapolation.

Method	S-NIAH-1				S-NIAH-2		
	2K	4K	8K	16K	2K	4K	8K
SWA only	26.8	–	6.8	2.8	–	–	–
LaCT	**100.0**	81.0	37.4	3.0	91.4	80.6	5.6
LaCT-Matched	99.8	**99.2**	51.6	0.0	**98.6**	65.8	4.8
Chunk-averaged	25.2	10.8	6.8	3.0	30.4	18.2	7.2
E^2-TTT$_{SwiGLU}$	95.6	95.0	**95.4**	**93.6**	96.8	**87.6**	**40.6**

5.5 Multimodal Evaluation

To assess generalization beyond language modeling, we integrate E^2-TTT into Qwen3VL-2B-Instruct [5] as a parallel branch fused with self-attention layers. We freeze the pre-trained backbone and train only the TTT parameters on a 43K-sample subset of LLaVA-Video-178K [67]. Qwen3VL+E^2-TTT improves over the frozen base model (43.9 vs. 41.9 on VideoMMMU, 59.0 vs. 56.9 on LongVideoBench) and matches a fully fine-tuned baseline trained on the same data (43.5 and 58.3, respectively). Additional training and evaluation details are provided in Appendix B.1 and B.2.

Table 6: Video understanding benchmarks. Integrating E^2-TTT into Qwen3VL-2B-Instruct and training only the TTT parameters matches full fine-tuning of the backbone on the same training data.

Model	VideoMMMU				LongVideoBench		
	Perc.	Comp.	Adap.	Avg.	Perc.	Rel.	Avg.
Qwen3VL	–	–	–	41.9	63.1	51.5	56.9
Qwen3VL (SFT)	60.3	**37.3**	33.0	43.5	63.4	54.0	58.3
Qwen3VL+E^2-TTT	**60.7**	**37.3**	**33.7**	**43.9**	**63.7**	**54.9**	**59.0**

6 Related Work

Test-Time Training (TTT) [52] is an emerging paradigm in sequence modeling that redefines the recurrent states in RNNs as learnable weights of an online-adapted non-linear neural network. These weights, often referred to as fast weights [49], are continuously updated to learn in-context. Existing methods typically employ a self-supervised loss that encourages these fast weights to memorize key-value associations from in-context tokens, effectively treating inference as an online learning problem using variants of gradient descent. While TTT [52, 57] has motivated a range of new recurrent architectures [7, 30, 8, 53], its token-wise updates are inherently sequential, leading to poor hardware utilization and limiting scalability. Recent efforts like LaCT [66] have attempted to mitigate this by shifting from token-wise to chunk-wise updates, sacrificing intra-chunk dependency modeling to improve throughput. However, these methods often resort to simplified aggregation rules (e.g., flat averaging) that degrade the optimization fidelity. Our work builds upon this foundation but distinguishes itself by introducing E^2-TTT (Expressive and Efficient TTT), a formulation that reconciles the hardware efficiency of chunking with the mathematical precision of complex, token-wise update dynamics. Combining global recurrence with local attention is a proven strategy for efficient long-context modeling [36, 47, 15]. Following recent work [28, 66], we adopt a *within-layer* hybrid architecture that shares the key, value, and query vectors between local attention and global recurrence. This design pattern has been successfully deployed in various forms [26, 41, 2]. However, prior instantiations typically rely on standard linear attention [31, 49, 63, 28] or simplified TTT update rules [66]. Our work retains this established architectural template but replaces the recurrence module with our E^2-TTT, empirically showing superior results.

7 Conclusion

We presented E^2-TTT (Expressive and Efficient TTT), a method that reconciles the hardware efficiency of chunk-wise processing with the per-token expressivity of the update rule in Test-Time Training. By deriving two closed-form scalar kernels, we enabled the parallel execution of per-token recurrent dynamics without the chunk-level approximations of prior chunk-wise methods. Empirically, E^2-TTT is on par with strong sub-quadratic baselines in language modeling and outperforms them on retrieval while demonstrating superior length extrapolation, sustaining $> 90\%$ passkey retrieval accuracy on "Needle in a Haystack" at $8\times$ the training context length, where baseline methods collapse. These results suggest that preserving precise temporal dynamics is essential for long-context generalization, positioning E^2-TTT as a promising direction for scalable Test-Time Training.

Acknowledgements

This work was supported by the JuBot project funded by the Carl-Zeiss-Foundation. The authors acknowledge support by the state of Baden-Württemberg through bwHPC. We acknowledge EuroHPC Joint Undertaking for awarding us access to MareNostrum5 at BSC, Spain. Experiments were performed on the HoreKa supercomputer, funded by the Ministry of Science, Research and the Arts Baden-Württemberg and by the Federal Ministry of Education and Research, and on MareNostrum5. Juergen Gall has been supported by the Federal Ministry of Research, Technology and Space (BMFTR) under grant no. 16IS22094A WEST-AI and the ERC Consolidator Grant FORHUE (101044724).

References

[1] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022. 18

[2] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. Language models enable simple systems for generating structured views of heterogeneous data lakes. *arXiv preprint arXiv:2304.09433*, 2023. 7, 10, 18

[3] Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff. *arXiv preprint arXiv:2402.18668*, 2024. 5

[4] Simran Arora, Aman Timalsina, Aaryan Singhal, Benjamin Spector, Sabri Eyuboglu, Xinyi Zhao, Ashish Rao, Atri Rudra, and Christopher Ré. Just read twice: closing the recall gap for recurrent language models. *arXiv preprint arXiv:2407.05483*, 2024. 18

[5] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025. 9, 19

[6] Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 3119–3137, 2024. 8, 18

[7] Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. *arXiv preprint arXiv:2501.00663*, 2024. 2, 3, 4, 10, 17, 21

[8] Ali Behrouz, Zeman Li, Praneeth Kacham, Majid Daliri, Yuan Deng, Peilin Zhong, Meisam Razaviyayn, and Vahab Mirrokni. Atlas: Learning to optimally memorize the context at test time. *arXiv preprint arXiv:2505.23735*, 2025. 2, 10

[9] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, pages 7432–7439, 2020. 18

[10] Loïc Cabannes, Pierre-Emmanuel Mazaré, Gergely Szilvasy, Matthijs Douze, Maria Lomeli, Ilze Amanda Auzina, Justin Carpentier, Gabriel Synnaeve, and Hervé Jégou. Sparse delta memory: Scaling the state of linear RNNs through sparsity. *arXiv preprint arXiv:2607.07386*, 2026. 1

[11] Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018. 18

[12] Karan Dalal, Daniel Koceja, Jiarui Xu, Yue Zhao, Shihao Han, Ka Chun Cheung, Jan Kautz, Yejin Choi, Yu Sun, and Xiaolong Wang. One-minute video generation with test-time training. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 17702–17711, 2025. 2, 18

[13] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv:2405.21060*, 2024. 1, 22

[14] Pradeep Dasigi, Kyle Lo, Iz Beltagy, Arman Cohan, Noah A Smith, and Matt Gardner. A dataset of information-seeking questions and answers anchored in research papers. *arXiv preprint arXiv:2105.03011*, 2021. 18

[15] Soham De, Samuel L Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, et al. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024. 10

[16] Guanting Dong, Xiaoshuai Song, Yuyang Hu, Jiajie Jin, Chenghao Zhang, Yifei Chen, Xiaoxi Li, Huaying Yuan, Xinyu Yang, Tongyu Wen, Jiejun Tan, Hongjin Qian, Shijue Huang, Junting Lu, Zhenyu Li, Wanjun Zhong, Yutao Zhu, Tat-Seng Chua, Zhicheng Dou, and Ji-Rong Wen. Towards long-horizon agents: A survey. *Preprints*, 2026. 1

[17] Alexander Richard Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir Radev. Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pages 1074–1084, 2019. 18

[18] Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, et al. A framework for few-shot language model evaluation. *Zenodo*, 2021. 18

[19] Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. Samsum corpus: A human-annotated dialogue dataset for abstractive summarization. *arXiv preprint arXiv:1911.12237*, 2019. 18

[20] 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. 6

[21] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First conference on language modeling*, 2024. 1, 18

[22] Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. Longcoder: A long-range pre-trained language model for code completion. In *International Conference on Machine Learning*, pages 12098–12107. PMLR, 2023. 18

[23] Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*, 2020. 18

[24] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024. 8, 18

[25] Kairui Hu, Penghao Wu, Fanyi Pu, Wang Xiao, Yuanhan Zhang, Xiang Yue, Bo Li, and Ziwei Liu. Video-mmmu: Evaluating knowledge acquisition from multi-discipline professional videos. *arXiv preprint arXiv:2501.13826*, 2025. 19

[26] Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. Transformer quality in linear time. In *International conference on machine learning*, pages 9099–9117. PMLR, 2022. 5, 10

[27] Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. *arXiv preprint arXiv:2104.02112*, 2021. 18

[28] Kazuki Irie, Morris Yau, and Samuel J Gershman. Blending complementary memory systems in hybrid quadratic-linear transformers. *arXiv preprint arXiv:2506.00744*, 2025. 6, 7, 10, 18, 19, 23

[29] Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551*, 2017. 18

[30] Mahdi Karami, Razvan Pascanu, and Vahab Mirrokni. Lattice: Learning to efficiently compress the memory. *arXiv preprint arXiv:2504.05646*, 2025. 10

[31] 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. 1, 10

[32] Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, Wentao Li, Enzhe Lu, Weizhou Liu, Yanru Chen, Weixin Xu, Longhui Yu, Yejie Wang, Yu Fan, Longguang Zhong, Enming Yuan, Dehao Zhang, Yizhi Zhang, T. Y. Liu, Haiming Wang, Shengjun Fang, Weiran He, Shaowei Liu, Yiwei Li, Jianlin Su, Jiezhong Qiu, Bo Pang, Junjie Yan, Zhejun Jiang, Weixiao Huang, Bohong Yin, Jiacheng You, Chu Wei, Zhengtao Wang, Chao Hong, Yutian Chen, Guanduo Chen, Yucheng Wang, Huabin Zheng, Feng Wang, Yibo Liu, Mengnan Dong, Zheng Zhang, Siyuan Pan, Wenhao Wu, Yuhao Wu, Longyu Guan, Jiawen Tao, Guohong Fu, Xinran Xu, Yuzhi Wang, Guokun Lai, Yuxin Wu, Xinyu Zhou, Zhilin Yang, and Yulun Du. Kimi linear: An expressive, efficient attention architecture. *arXiv:2510.26692*, 2025. 1

[33] Tomáš Kočiskỳ, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. The narrativeqa reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018. 18

[34] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023. 1

[35] Xin Li and Dan Roth. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002. 18

[36] Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, et al. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024. 10

[37] Tianyang Liu, Canwen Xu, and Julian McAuley. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*, 2023. 18

[38] Colin Lockard, Prashant Shiralkar, and Xin Luna Dong. Openceres: When open information extraction meets the semi-structured web. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3047–3056, 2019. 7, 18

[39] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 6

[40] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016. 18

[41] Tsendsuren Munkhdalai, Manaal Faruqui, and Siddharth Gopal. Leave no context behind: Efficient infinite context transformers with infini-attention. *arXiv preprint arXiv:2404.07143*, 2024. 5, 10

[42] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 1525–1534, 2016. 18

[43] Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin A Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. *Advances in Neural Information Processing Systems*, 37:30811–30849, 2024. 2, 6

[44] Qwen Team. Qwen3-next: Towards ultimate training & inference efficiency, 2025. 1

[45] Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019. 18

[46] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018. 7, 18

[47] Liliang Ren, Yang Liu, Yadong Lu, Yelong Shen, Chen Liang, and Weizhu Chen. Samba: Simple hybrid state space models for efficient unlimited context language modeling. *arXiv preprint arXiv:2406.07522*, 2024. 10

[48] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 8732–8740, 2020. 18

[49] 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. 1, 2, 10, 19

[50] Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*, 2022. 3

[51] Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. *Advances in Neural Information Processing Systems*, 37:7339–7361, 2024. 6

[52] 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. 1, 2, 3, 6, 10, 17, 21, 23

[53] Arnuv Tandon, Karan Dalal, Xinhao Li, Daniel Koceja, Marcel Rød, Sam Buchanan, Xiaolong Wang, Jure Leskovec, Sanmi Koyejo, Tatsunori Hashimoto, Carlos Guestrin, Jed McCaleb, Yejin Choi, and Yu Sun. End-to-end test-time training for long context. *arXiv:2512.23675*, 2025. 2, 10, 22

[54] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. 6, 7

[55] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. 18

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017. 1

[57] 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. 1, 10

[58] Haoning Wu, Dongxu Li, Bei Chen, and Junnan Li. Longvideobench: A benchmark for long-context interleaved video-language understanding. *Advances in Neural Information Processing Systems*, 37: 28828–28857, 2024. 19

[59] Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, et al. Effective long-context scaling of foundation models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 4643–4663, 2024. 8

[60] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, 2025. 1

[61] 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. 6

[62] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024. 1, 2, 4, 18

[63] Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. *Advances in neural information processing systems*, 37:115491–115522, 2024. 1, 6, 7, 10, 18

[64] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 2369–2380, 2018. 18

[65] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019. 18

[66] Tianyuan Zhang, Sai Bi, Yicong Hong, Kai Zhang, Fujun Luan, Songlin Yang, Kalyan Sunkavalli, William T Freeman, and Hao Tan. Test-time training done right. *arXiv preprint arXiv:2505.23884*, 2025. 2, 3, 5, 6, 7, 10, 17, 18, 19, 21, 22

[67] Yuanhan Zhang, Jinming Wu, Wei Li, Bo Li, Zejun Ma, Ziwei Liu, and Chunyuan Li. Video instruction tuning with synthetic data. *arXiv preprint arXiv:2410.02713*, 2024. 9, 18

[68] Ming Zhong, Da Yin, Tao Yu, Ahmad Zaidi, Mutethia Mutuma, Rahul Jha, Ahmed Hassan, Asli Celikyilmaz, Yang Liu, Xipeng Qiu, et al. Qmsum: A new benchmark for query-based multi-domain meeting summarization. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5905–5921, 2021. 18

A Detailed Derivation of the Parallel Kernel

In this section, we derive the closed-form expansion of the TTT update rule within a single chunk of length C. This formulation allows us to "jump" directly from the start of the chunk ($t = 0$) to the end ($t = C$) without materializing intermediate states. The chunk index r is omitted for simplicity.

Recall that we consider general per-token updates with decoupled weight decay and momentum. Let the fast weights $\mathbf{W}_t \in \mathbb{R}^{d\times d}$, momentum $\mathbf{M}_t \in \mathbb{R}^{d\times d}$, and update direction (negative gradient) $\mathbf{G}_t \in \mathbb{R}^{d\times d}$ be matrix-valued. Let $\gamma_t, \beta_t \in (0, 1)$ be the scalar decay and momentum factors, and $\eta_t > 0$ be the learning rate. The recurrence is:

$$\mathbf{M}_t = \beta_t \mathbf{M}_{t-1} + \eta_t \mathbf{G}_t, \qquad \mathbf{W}_t = \gamma_t \mathbf{W}_{t-1} + \mathbf{M}_t. \tag{17}$$

A.1 Unrolling Momentum and Weights

Unrolling the momentum. By induction, the momentum $\mathbf{M}_t$ at step t is the sum of the decayed initial momentum at the chunk start and the history of gradients:

$$\mathbf{M}_t = \Big(\prod_{i=1}^{t} \beta_i\Big)\mathbf{M}_0 + \sum_{j=1}^{t} \Big(\eta_j \mathbf{G}_j \prod_{i=j+1}^{t} \beta_i\Big). \tag{18}$$

Unrolling the Weights. Similarly, unrolling the weight update (Eq. 17) yields:

$$\mathbf{W}_t = \Big(\prod_{i=1}^{t} \gamma_i\Big)\mathbf{W}_0 + \sum_{j=1}^{t} \Big(\mathbf{M}_j \prod_{i=j+1}^{t} \gamma_i\Big). \tag{19}$$

Insert (18) into (19) and separate the $\mathbf{M}_0$ and gradient parts:

$$\mathbf{W}_t = \underbrace{\Big(\prod_{i=1}^{t} \gamma_i\Big)\mathbf{W}_0}_{\text{Decay term}} + \underbrace{\sum_{j=1}^{t} \Big(\Big(\prod_{i=1}^{j} \beta_i\Big)\mathbf{M}_0 \prod_{q=j+1}^{t} \gamma_q\Big)}_{\text{Momentum term}} + \underbrace{\sum_{j=1}^{t}\sum_{k=1}^{j} \Big(\eta_k \mathbf{G}_k \prod_{i=k+1}^{j} \beta_i \prod_{q=j+1}^{t} \gamma_q\Big)}_{\text{Gradient term}}. \tag{20}$$

Re-indexing the gradient contribution. Swap the order of summation in the last term of (20) by fixing the gradient index k and summing over $j = k, \ldots, t$:

$$\sum_{j=1}^{t}\sum_{k=1}^{j} \Big(\eta_k \mathbf{G}_k \prod_{i=k+1}^{j} \beta_i \prod_{q=j+1}^{t} \gamma_q\Big) = \sum_{k=1}^{t} \eta_k \mathbf{G}_k \sum_{j=k}^{t} \Big(\prod_{i=k+1}^{j} \beta_i \prod_{q=j+1}^{t} \gamma_q\Big). \tag{21}$$

This converts Eq. 20 to:

$$\mathbf{W}_t = \underbrace{\Big(\prod_{i=1}^{t} \gamma_i\Big)\mathbf{W}_0}_{\text{Decay term}} + \underbrace{\sum_{j=1}^{t} \Big(\Big(\prod_{i=1}^{j} \beta_i\Big)\mathbf{M}_0 \prod_{q=j+1}^{t} \gamma_q\Big)}_{\text{Momentum term}} + \underbrace{\sum_{k=1}^{t} \eta_k \mathbf{G}_k \sum_{j=k}^{t} \Big(\prod_{i=k+1}^{j} \beta_i \prod_{q=j+1}^{t} \gamma_q\Big)}_{\text{Gradient term}}. \tag{22}$$

A.2 Closed form at the final step per chunk ($t = C$)

To efficiently compute the final state $\mathbf{W}_C$, we define three auxiliary scalar sequences. Let the suffix products for decay γ and momentum β be:

$$\tilde{\beta}_t := \prod_{i=t+1}^{C} \beta_i, \qquad \tilde{\gamma}_t := \prod_{i=t+1}^{C} \gamma_i, \tag{23}$$

(with the empty product convention $\prod_{C+1}^{C}(\cdot) = 1$). We further define the cumulative ratio sum R_t:

$$R_t := \sum_{i=t}^{C} \frac{\tilde{\gamma}_i}{\tilde{\beta}_i}. \tag{24}$$

Since these terms depend only on the scalars β, γ, they can be pre-computed efficiently in parallel (e.g., via log-space cumulative sums).

Initial Weight Carry ($\mathrm{carry_w}$)**.** Directly from definition:

$$\mathrm{carry_w} := \prod_{i=1}^{C} \gamma_i = \tilde{\gamma}_0 \tag{25}$$

Initial Momentum Carry ($\mathrm{carry_m}$)**.** We rewrite the prefix product $\prod_{i=1}^{j} \beta_i$ as the total product divided by the suffix product:

$$\prod_{i=1}^{j} \beta_i = \big(\prod_{i=1}^{C} \beta_i\big)/\tilde{\beta}_j. \tag{26}$$

Substituting this into the $\mathbf{M}_0$ term in Eq. 22:

$$\sum_{j=1}^{C} \Big(\big(\prod_{i=1}^{j} \beta_i\big)\, \mathbf{M}_0\, \tilde{\gamma}_j \Big) = \Big(\prod_{i=1}^{C} \beta_i\Big) \underbrace{\sum_{j=1}^{C} \frac{\tilde{\gamma}_j}{\tilde{\beta}_j}}_{=R_1} \mathbf{M}_0, \tag{27}$$

with

$$\mathrm{carry_m} := \tilde{\beta}_0 R_1. \tag{28}$$

Gradient contributions. From (21) at $t = C$, each $\mathbf{G}_k$ is weighted by the scalar

$$\sum_{j=k}^{C} \Big(\prod_{i=k+1}^{j} \beta_i \prod_{q=j+1}^{C} \gamma_q \Big) = \tilde{\beta}_k \sum_{j=k}^{C} \frac{\tilde{\gamma}_j}{\tilde{\beta}_j} = \tilde{\beta}_k R_k. \tag{29}$$

Closed form for $\mathbf{W}_C$. Combining these terms, the final weight matrix $\mathbf{W}_C$ is computed as:

$$\mathbf{W}_C = \tilde{\gamma}_0 \mathbf{W}_0 \; + \; \tilde{\beta}_0 R_1 \mathbf{M}_0 \; + \; \sum_{t=1}^{C} \big(\eta_t\, \tilde{\beta}_t\, R_t\big)\, \mathbf{G}_t. \tag{30}$$

Closed form for $\mathbf{M}_C$. Since the momentum recurrence does not depend on $\mathbf{W}_t$, its closed form follows directly from (18) by setting $t = C$:

$$\mathbf{M}_C = \Big(\prod_{i=1}^{C} \beta_i\Big)\mathbf{M}_0 + \sum_{t=1}^{C} \Big(\eta_t\, \mathbf{G}_t \prod_{i=t+1}^{C} \beta_i\Big). \tag{31}$$

Recognizing the suffix products defined in (23), $\tilde{\beta}_0 = \prod_{i=1}^{C} \beta_i$ and $\tilde{\beta}_t = \prod_{i=t+1}^{C} \beta_i$, Eq. (31) collapses to the parallel kernel form

$$\mathbf{M}_C = \tilde{\beta}_0\, \mathbf{M}_0 \; + \; \sum_{t=1}^{C} \big(\eta_t\, \tilde{\beta}_t\big)\, \mathbf{G}_t. \tag{32}$$

Unlike $\mathbf{W}_C$, the momentum kernel involves no cumulative ratio R_t: each gradient $\mathbf{G}_t$ is propagated to the chunk end through a single suffix product $\tilde{\beta}_t$, since only one decay channel (β) acts on $\mathbf{M}_t$.

This formulation allows us to "jump" directly from $t = 0$ to $t = C$ without materializing intermediate states. To ensure numerical stability, particularly with long chunk lengths C, we perform the cumulative product computations $(\tilde{\beta}, \tilde{\gamma})$ in log-space and convert back to linear space only for the final summation.

A.3 Stability of the Cumulative Kernel R_t

We analyze the magnitude of the cumulative kernel R_t under two regimes that bound the practical operating range of our input-dependent dynamics: (i) near-unit retention ($\beta_t \to 1$ and $\gamma_t \to 1$), as encouraged by the timescale parameterization (Eq. 13) for long-range memory, and (ii) aggressive decay (γ_t small).

Near-unit retention regime. When $\beta_t \to 1$ and $\gamma_t \to 1$ for all t, the suffix products satisfy $\tilde{\beta}_i \to 1$ and $\tilde{\gamma}_i \to 1$, so $\tilde{\gamma}_i/\tilde{\beta}_i \to 1$ for every i. Substituting into the definition of R_t gives

$$R_t \;\to\; \sum_{i=t}^{C} 1 \;=\; C - t + 1 \;\le\; C. \tag{33}$$

The weight kernel $\mathcal{K}_t^W = \eta_t \, \tilde{\beta}_t \, R_t$ is therefore bounded by

$$\mathcal{K}_t^W \;\le\; \eta_t \cdot 1 \cdot C. \tag{34}$$

Because η_t is sigmoid-gated and scaled by η_{base} (Eq. 12), the per-token kernel contribution is bounded by $\eta_{\text{base}} \cdot C$. With our default $\eta_{\text{base}} = 10^{-2}$ and $C = 512$, this yields a worst-case bound of 5.12. The momentum kernel $\mathcal{K}_t^M = \eta_t \, \tilde{\beta}_t$ is more tightly bounded by η_{base}.

Aggressive decay regime. When γ_t is small, the suffix product $\tilde{\gamma}_i$ decays geometrically as i moves away from the chunk end, and so does the ratio $\tilde{\gamma}_i / \tilde{\beta}_i$. The sum R_t is then dominated by its last few terms (i close to C) and remains $\mathcal{O}(1)$.

Coupled parameterization. The decay parameterization $\gamma_t = 1 - \eta_t \alpha_t$ (Eq. 14) enforces a structural coupling: large updates (η_t large) automatically increase forgetting (γ_t smaller), preventing carry inflation across chunks. This analysis is consistent with the numerical verification reported in Sec. 3.2: over a 128-chunk trajectory ($C = 512$, 65,536 tokens), the relative ℓ_2 deviation from the fully sequential reference is below 2×10^{-6} on both $\mathbf{W}_C$ and $\mathbf{M}_C$.

A.4 Single Backward Pass for Both Aggregates

Proposition 3.1 expresses both chunk-end states through a sum over per-token gradients $\mathbf{G}_t = -\nabla_{\mathbf{W}} \mathcal{L}_t \big|_{\mathbf{W}_0}$:

$$\sum_{t=1}^{C} \mathcal{K}_t^W \, \mathbf{G}_t \;=\; -\sum_{t=1}^{C} \mathcal{K}_t^W \, \nabla_{\mathbf{W}} \mathcal{L}_t \big|_{\mathbf{W}_0}, \tag{35}$$

$$\sum_{t=1}^{C} \mathcal{K}_t^M \, \mathbf{G}_t \;=\; -\sum_{t=1}^{C} \mathcal{K}_t^M \, \nabla_{\mathbf{W}} \mathcal{L}_t \big|_{\mathbf{W}_0}, \tag{36}$$

where the kernels $\mathcal{K}_t^W = \eta_t \, \tilde{\beta}_t \, R_t$ and $\mathcal{K}_t^M = \eta_t \, \tilde{\beta}_t$ differ only by the scalar factor R_t.

Chain-rule decomposition. Each per-token loss has the form $\mathcal{L}_t = \mathcal{L}(f_{\mathbf{W}_0}(\boldsymbol{k}_t), \boldsymbol{v}_t)$. Applying the chain rule,

$$\nabla_{\mathbf{W}} \mathcal{L}_t \big|_{\mathbf{W}_0} \;=\; -\left(\frac{\partial f_{\mathbf{W}_0}(\boldsymbol{k}_t)}{\partial \mathbf{W}} \right)^{\top} \boldsymbol{g}_t, \qquad \boldsymbol{g}_t \;:=\; -\frac{\partial \mathcal{L}_t}{\partial f_{\mathbf{W}_0}(\boldsymbol{k}_t)}. \tag{37}$$

The Jacobian $(\partial f_{\mathbf{W}_0}(\boldsymbol{k}_t) / \partial \mathbf{W})^{\top}$ depends only on $(\mathbf{W}_0, \boldsymbol{k}_t)$ and is independent of the scalar kernels.

Shared backward, scalar re-weighting. Both aggregates therefore reduce to scalar-weighted contractions over the same set of activation gradients $\{\boldsymbol{g}_t\}_{t=1}^{C}$ and per-token inputs $\{\boldsymbol{k}_t\}_{t=1}^{C}$, differing only in the scalar weight $\mathcal{K}_t^W$ versus $\mathcal{K}_t^M$ applied per token. A single autograd backward pass through the per-token losses produces $\{\boldsymbol{g}_t\}$. Both aggregates are then constructed by reusing these intermediates with the appropriate scalar weighting. The marginal cost of producing $\mathbf{M}_C$ in addition to $\mathbf{W}_C$ is therefore one extra scalar-weighted contraction, not a second backward pass. This is consistent with the throughput measurements reported in Fig. 5, where E^2-TTT stays close to the throughput of LaCT [66] despite computing the exact closed-form kernel.

A.5 Two Equivalent Views of the Chunk Dynamics

Within a chunk, all gradients $\mathbf{G}_t$ in our formulation are evaluated at the frozen $\mathbf{W}_0$ (Sec. 3.2). Under this constraint, the closed-form chunk-end states from Proposition 3.1 admit two complementary readings.

Optimization-style view. Equation (7) reads as a token-wise state recurrence with momentum factor β_t and L$_2$ decay factor γ_t, both modulated per token. This is the framing we adopt throughout the paper, for consistency with the mini-batch and chunk-wise TTT literature [52, 7, 66]. This view is accurate at the inter-chunk level, where $\mathbf{M}_C$ is carried into the next chunk and accumulates momentum over the full sequence.

Kernel-weighting view. Equivalently, the chunk-end states can be read as scalar-weighted aggregations of per-token gradients,

$$\mathbf{W}_C \;=\; \tilde{\gamma}_0 \mathbf{W}_0 + \tilde{\beta}_0 R_1 \mathbf{M}_0 + \sum_{t=1}^{C} \mathcal{K}_t^W \, \mathbf{G}_t, \qquad \mathbf{M}_C \;=\; \tilde{\beta}_0 \mathbf{M}_0 + \sum_{t=1}^{C} \mathcal{K}_t^M \, \mathbf{G}_t,$$

in which $(\mathcal{K}_t^W, \mathcal{K}_t^M)$ act as learned, time-aware kernel weights over per-token gradient contributions to the chunk-end states.

The two views are mathematically equivalent for the chunk-end pair $(\mathbf{W}_C, \mathbf{M}_C)$. They differ only in emphasis. The optimization-style view is the natural reading across chunk boundaries, where momentum is carried forward. The kernel-weighting view is the natural reading within a chunk, where the frozen-$\mathbf{W}_0$ constraint makes "momentum" and "decay" equivalent to a learned per-token weighting of gradient contributions.

B Experiment Continued

B.1 Training Details

Table 7 shows the training and model hyperparameters used to train the 340M and 1.3B parameter models. We train with an effective batch size of 256 and a sequence length of 2048 for 28,672 steps, amounting to 15B tokens. The 1.3B models are trained with a slightly increased sequence length of 2240, which increases the training token count to 16B (for simplicity, in the main text, we refer to both as trained for 15B tokens). We implement our expressive chunk-wise TTT in plain PyTorch without custom Triton kernels. Training the 340M and 1.3B models takes about 132 and 348 total GPU hours on H100 GPUs, respectively. Following Zhang et al. [66], we apply L2 normalization to the fast-weight matrices at each chunk boundary as a standard stabilizer.

Table 7: Model hyperparameters.

	Model	
	340M	1.3B
Number of layers	24	
Feedforward block multiplier	4	
Total hidden size	1024	2048
Number of heads	8	16
Sequence length	2048	2240
Effective Batch size	256	
Learning rate	10^{-3}	
Warmup steps	1024	
Minimum learning rate ratio	0.1	
Max norm clipping	1.0	
Std. of weight initializers	0.02	
TTT base learning rate η_{base}	0.01	
TTT momentum timescale τ	32	
TTT decay strength α_{base}	0.1	

In our video understanding experiments, we instantiate E^2-TTT (parameterized by a SwiGLU MLP) as a parallel branch fused with the existing Qwen3VL self-attention layers. E^2-TTT employs its own query, key, and value projection layers. Following standard practice [1, 12], we gate the output of E^2-TTT with a learned vector $\varepsilon \in \mathbb{R}^d$ with tanh activation. All entries of ε are initialized to 0.1, such that $\tanh(\varepsilon)$ is close to 0 at the beginning of fine-tuning. We train only the TTT parameters on a 43K-sample subset of LLaVA-Video-178K [67].

B.2 Evaluation Details

Commonsense Reasoning. Following prior work [21, 62, 28], we evaluate our model on multiple commonsense reasoning benchmarks: PIQA [9], HellaSwag [Hella.; 65], WinoGrande [Wino.; 48], ARC-easy (ARC-e) and ARC-challenge (ARC-c) [11], Wikitext [Wiki.; 40], and LAMBADA [LMB.; 42]. All evaluations are performed using `lm-evaluation-harness` [18].

In-context Retrieval. Following prior work [63, 2, 28], we focus on three tasks: FDA [2], SWDE [38], and SQuAD [46]. For SQuAD and SWDE, we use `lm-evaluation-harness` [18] for evaluation. For FDA, we follow Irie et al. [28] and use the evaluation script from Arora et al. [4].

Length Extrapolation. We first assess whether the models can maintain stable language modeling performance when processing sequences longer than their training context (2K tokens). Following Yang et al. [62], we conduct experiments on six diverse long-context benchmarks: PG19 [45], GovReport [27], QMSum [68], NarrativeQA [33], Qasper [14], and CodeParrot. To test effective capacity, we utilize the S-NIAH-1 (passkey retrieval) and S-NIAH-2 (numerical needle in haystack) tasks from RULER's Needle-In-A-Haystack Single (NIAH-S) benchmark suite [24]. Finally, we evaluate on 14 real-world long-context tasks from Longbench [6], including: narrative comprehension (Narrative QA [33]), scientific understanding (QasperQA [14]), multi-hop reasoning (MultiField QA, HotpotQA [64], 2WikiMulti QA [23], Musique [55]), document summarization (GovReport [27], QMSum [68], MultiNews [17]), and various specialized tasks (TRec [35], Trivia QA [29], SamSum [19], LCC [22], and RepoBench-P [37]). All evaluations, except for the stability analysis, are performed using `lm-evaluation-harness` [18].

Video Understanding. We evaluate on two standard video understanding benchmarks: VideoMMMU [25] and LongVideoBench [58]. Following the Qwen3VL [5] protocol, we employ a uniform sampling strategy to extract a fixed sequence of frames from each video. We configure the frame count and token budget to balance temporal range with spatial resolution: for VideoMMMU, we sample 512 frames with a maximum of 256 tokens per frame, and for LongVideoBench, 256 frames with 512 tokens.

Hybrid Baselines. We benchmark against two competitive hybrid architectures: HQLT [28] and LaCT [66]. For HQLT, which combines DeltaNet [49] with window attention, we select the *synchronous* variant where both components process inputs simultaneously, which the authors report as the best-performing setting. For LaCT, which fuses chunk-wise TTT (parameterized by a SwiGLU MLP) with window attention, we evaluate the variant using Muon optimization for the TTT update rule, consistent with the best-performing configuration provided in the official implementation.

B.3 Ablation Study

We validate the design of E^2-TTT along four axes. **(i)** The proposed closed-form kernel is the source of accuracy gains over chunk-wise baselines that average per-token factors, and finer chunk granularity narrows the residual gap to the ideal token-wise update (Table 8). **(ii)** Both branches of the hybrid architecture and the post-chunk normalization are individually necessary, as removing any one of them collapses either retrieval or long-context performance (Table 9). **(iii)** The token-wise momentum and weight-decay terms recovered exactly by our kernel reduce perplexity over a no-momentum baseline (Table 10). **(iv)** The method is robust to the choice of base learning rate and base decay over a wide range (Table 11).

Table 8: Effect of the closed-form kernel and chunk granularity (340M parameters, 15B tokens). The averaged-factor baseline collapses per-token β_t, γ_t into chunk-level scalars, whereas our kernel preserves the full per-token dynamics in closed form.

Configuration	Wiki ppl ↓	LMB ppl ↓	Avg ↑
Averaged-factor baseline (C=512)	26.9	25.9	48.6
Closed-form kernel, C=1024	25.6	26.2	48.4
Closed-form kernel, C=512	**25.5**	**25.0**	**49.1**

Effect of the closed-form kernel and chunk granularity (Table 8). We compare against a chunk-wise baseline that averages the per-token decay and momentum factors into chunk-level scalars, following the simplification used in LaCT [66]. Replacing the averaged factors with our closed-form kernel reduces Wiki perplexity from 26.9 to 25.5 and improves average accuracy from 48.6% to 49.1% at C=512. We then sweep the chunk size: C=1024 already outperforms the averaged baseline (Wiki ppl 25.6, Avg 48.4%), and C=512 further improves both metrics. This trend matches the theoretical prediction that $C \to 1$ recovers the token-wise update exactly. Smaller chunks narrow the gap but reduce hardware utilization. We therefore use C=512 in the main experiments as a trade-off between efficiency and quality.

Table 9: Component ablation at the 1.3B scale on S-NIAH-1 (accuracy ↑ at three context lengths) and LongBench (average score ↑). The full model is trained for 15B tokens at 2K context. All ablations use the same training data and hyperparameters.

Configuration	S-NIAH-1 2K	S-NIAH-1 8K	S-NIAH-1 16K	LongBench
SWA only (no TTT branch)	26.8	6.8	2.8	11.2
TTT only (no SWA branch)	0.0	0.0	0.0	1.5
Full, w/o post-chunk norm	**100.0**	**99.8**	**98.8**	13.4
Full, w/o fusion gate	99.6	88.8	86.6	13.3
Full E^2-TTT$_{\text{SwiGLU}}$	95.6	95.4	93.6	**14.1**

Branch contributions (Table 9). We isolate the role of each component of the hybrid architecture at the 1.3B scale, evaluating on Single-NIAH-1 at three context lengths and on LongBench. Sliding-window attention alone reaches only 6.8% on S-NIAH-1 at 8K tokens, confirming that local attention cannot solve long-range retrieval. The TTT branch alone (no SWA) collapses to 0.0% across all

S-NIAH lengths, because the chunk-wise output step cannot resolve within-chunk dependencies. This is the "blind spot" that our hybrid design addresses. With both branches, S-NIAH-1 reaches 95.4% at 8K and 93.6% at 16K. Post-chunk normalization trades a small amount of synthetic-retrieval accuracy (99.8 versus 95.4 on S-NIAH-1 at 8K) for real-world long-context performance (LongBench 14.1 versus 13.4). Removing the fusion gate costs accuracy beyond 2K on both axes.

Table 10: Effect of the inner-loop optimizer terms recovered by our kernel (340M, C=512, 15B tokens). The baseline retains only the input-dependent learning rate η_t.

Inner-loop update rule	Wiki ppl ↓	LMB ppl ↓	Avg ↑
η_t only (no momentum, no decay)	26.6	26.4	48.7
η_t + momentum + weight decay	**25.5**	**25.0**	**49.1**

Inner-loop momentum and weight decay (Table 10). We test whether the input-dependent momentum and decay terms contribute, by training two variants with C=512: a no-momentum/no-decay baseline (which retains only the input-dependent learning rate) and the full update rule. Adding momentum and weight decay reduces perplexity on both Wiki and LMB.

Table 11: Sensitivity of E^2-TTT$_{\text{MLP}}$ (340M) to the base hyperparameters of the input-dependent dynamics.

Configuration	Wiki ppl ↓	LMB ppl ↓	Avg ↑
Base learning rate η_{base} (α_{base}=10^{-1})			
$\eta_{\text{base}} = 10^{-1}$	28.8	30.3	46.5
$\eta_{\text{base}} = 10^{-2}$ *(default)*	25.5	25.0	49.1
$\eta_{\text{base}} = 10^{-3}$	26.1	23.0	49.1
Base decay α_{base} (η_{base}=10^{-2})			
$\alpha_{\text{base}} = 10^{-1}$ *(default)*	25.5	25.0	49.1
$\alpha_{\text{base}} = 10^{-2}$	25.8	23.6	49.0
$\alpha_{\text{base}} = 10^{-3}$	25.7	22.9	49.0

Sensitivity to base hyperparameters (Table 11). The input-dependent dynamics expose a base learning rate η_{base} and a base decay α_{base} that scale the per-token sigmoid gates. We sweep each over two orders of magnitude. Performance is essentially invariant to α_{base} in the tested range and stable for $\eta_{\text{base}} \in \{10^{-3}, 10^{-2}\}$. Only the aggressive η_{base}=10^{-1} degrades training, consistent with classical optimization where overly large step sizes destabilize updates. The defaults η_{base}=10^{-2} and α_{base}=10^{-1} used throughout the paper are not finely tuned.

B.4 Analysis of the Learned Coefficients

The ablations of Appendix B.3 remove a term from the update rule and retrain. A complementary question is whether the three linear heads of Eqs. (12)–(14) actually use their input-dependence once trained, or whether they instead settle near constants or silently compensate for one another. We answer this with two measurements that require no retraining: both run post-hoc on the single trained 1.3B E^2-TTT$_{\text{SwiGLU}}$ checkpoint used throughout the main experiments.

The coefficients are not near-constant. We measure dispersion on the three positive quantities whose zero value means "no effect": the step size η_t, the forgetting rate $-\log \beta_t$, and the decay rate $1-\gamma_t = \eta_t \alpha_t$. Over WikiText-2, the median coefficient of variation across all (layer, head) pairs is 0.95 for η_t, 0.46 for $-\log \beta_t$, and 1.29 for $1-\gamma_t$. Fewer than 1% of pairs fall below 0.1. Compounded over a chunk of C=512 tokens, the spread is substantial: the fast-weight carry $\prod_t \gamma_t$ spans 0.73–1.00 and the momentum carry $\prod_t \beta_t$ spans a factor of 129 between its 5th and 95th percentiles. This is the per-token variation that a chunk-averaged update rule discards.

Causal test: shuffling the coefficients along the token axis. A wide distribution alone does not establish that the values are aligned with the tokens that produced them. For each head we therefore permute its scalar along the token axis, independently per sequence and per head. The permutation preserves the marginal distribution exactly and destroys only the correspondence between a coefficient and its token. Any degradation can therefore be attributed to input-dependence alone. For α we

permute the head output and recompute $\gamma_t = 1 - \eta_t \alpha_t$, which leaves η_t aligned. All variants are scored with the same `lm-evaluation-harness` invocation used for Table 2.

Table 12: Shuffling each coefficient head along the token axis on the trained 1.3B E^2-TTT_{SwiGLU} checkpoint. The permutation leaves the marginal distribution of each scalar unchanged and removes only its alignment with the input.

Shuffled head	Wiki ppl ↓	Δ ppl	S-NIAH-1 16K ↑
— (baseline)	**19.94**	—	**93.6**
η	20.39	+0.45	6.0
β	20.05	+0.11	86.2
α	19.99	+0.05	91.8
All three	20.54	+0.60	4.0

Every shuffle degrades performance, and η has by far the largest effect. Shuffling η alone costs $+0.45$ perplexity and collapses retrieval ($93.6 \rightarrow 6.0$ on S-NIAH-1 at 16K), confirming that the step size carries most of the write-selectivity of the inner loop. The joint intervention costs approximately the sum of the individual ones ($+0.60$ versus $0.45 + 0.11 + 0.05$), indicating that the three heads contribute independently. If the heads substituted for one another, shuffling one alone would be cheap because the other two would absorb its role, and the joint cost would exceed the sum. This additivity can only be assessed on perplexity, since the η shuffle alone already drives S-NIAH accuracy to its floor.

The β and α shuffles have smaller effects than the η shuffle partly by construction. Because Eq. (14) couples the decay to the step size, permuting only the α head leaves the η-driven component of γ_t aligned. Nonetheless, Table 10 independently shows that the momentum and decay terms matter, as removing them from the update rule increases Wiki perplexity from 25.5 to 26.6 at 340M. Taken together, the two measurements show that the per-token scalars are input-dependent and non-redundant, which supports aggregating them exactly with the closed-form kernel (Sec. 3.2) instead of averaging them per chunk.

B.5 Throughput and Inference Cost

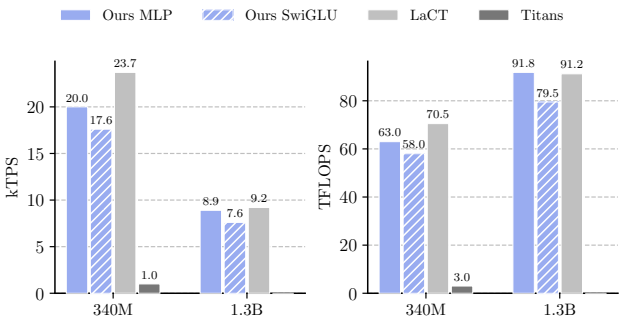

Figure 5: Training throughput on a single H100 GPU at the 340M and 1.3B scales.

To validate the hardware efficiency of our proposed method, we evaluate training throughput (measured in kilo-tokens per second, kTPS) and utilization (TFLOPS) on a single H100 64GB GPU. We test two model scales (340M and 1.3B parameters) on 2K sequences, using batch sizes of 4 and 2, respectively. We compare E^2-TTT against Titans [7] and LaCT [66]. All baselines are implemented in PyTorch under identical training conditions. As no official Titans implementation was available at the time of writing, we report a self-reproduction following the architecture described in Behrouz et al. [7]. We implemented both the scan-based formulation and the memory-optimized dual form [52] and report the latter, which we found slightly faster in PyTorch. As shown in Figure 5, Titans is severely throughput-bound: at 340M it achieves less than 2 kTPS, and at 1.3B it goes out-of-memory. This is consistent with the structure of the dual form for nonlinear inner models such as TTT-MLP: each inner-model layer is reformulated as a masked attention-like computation (Appendix A.3, Sun et al. [52]), and this machinery lies on the outer-loop autograd path, inflating both activation memory and

backward-pass cost. The dual form does eliminate per-token state materialization in principle, but for TTT-MLP the small inner mini-batch size (64 in our setting) combined with these masked-matmul overheads dominates throughput. These costs are absent under chunk-wise output (Eq. 6), where the per-chunk forward and backward operate on a single fast-weight state. In contrast, both E^2-TTT variants stay close to the throughput of the highly optimized LaCT baseline despite computing the exact closed-form kernel rather than a single averaged factor. Our kernel adds log-space suffix products, ratio sums, and a scalar re-weighting that yields $\mathbf{M}_C$ from the same backward pass that produces $\mathbf{W}_C$. This bookkeeping is cheap relative to the fast-weight forward and backward passes. Relative to LaCT, the exact kernel costs 15.6% throughput at 340M and only 3.3% at 1.3B for E^2-TTT$_{\text{MLP}}$. The overhead shrinks with scale because the d^2 fast-weight forward and backward passes increasingly dominate the scalar bookkeeping. Exact per-token dynamics therefore come at a small cost in training throughput.

Table 13: Inference cost at 1.3B on a single H100 GPU, at batch sizes 1 and 8. Decode latency is the mean per-token time over 512 steps after warm-up (median of three runs). All recurrent models are flat in both latency and memory between 2K and 16K, while attention is not.

	Decode (ms/token) ↓		Peak mem. (GB) ↓	
Model	2K	16K	2K	16K
Batch size 1				
Transformer++	14.8	15.0	3.25	6.10
Titans	56.1	56.2	3.41	3.51
LaCT	36.8	36.5	3.33	3.42
E^2-TTT$_{\text{MLP}}$	33.8	33.6	3.43	3.52
E^2-TTT$_{\text{SwiGLU}}$	34.0	33.9	3.62	3.71
Batch size 8				
Transformer++	14.8	59.9	7.50	29.90
Titans	56.5	56.6	6.40	6.40
LaCT	38.2	38.8	7.10	7.10
E^2-TTT$_{\text{MLP}}$	33.7	33.8	6.86	6.86
E^2-TTT$_{\text{SwiGLU}}$	34.3	34.4	8.08	8.08

Inference cost. Table 13 reports decode latency and peak memory at 1.3B. Attention is faster at short sequences thanks to its optimized kernels, but it is the only model whose cost grows with length ($14.8 \rightarrow 59.9$ ms/token and $7.50 \rightarrow 29.90$ GB between 2K and 16K at batch 8), while all recurrent models are flat in both because their state is of constant size. Among TTT methods, Titans is the slowest to decode, for the same reason it is slow to train. Both E^2-TTT variants decode slightly faster than LaCT at comparable memory, as the per-chunk scalar bookkeeping is amortized over C tokens and adds negligible cost to the fast-weight forward pass. The one exception is the peak memory of E^2-TTT$_{\text{SwiGLU}}$ at batch 8 (8.08 against LaCT's 7.10 GB), which comes from its three-matrix fast-weight network and the extra re-weighted aggregation (Algorithm 1). E^2-TTT$_{\text{MLP}}$, whose fast-weight network is smaller, stays below LaCT at batch 8 (6.86 versus 7.10 GB) and within 0.1 GB of it at batch 1.

B.6 Detailed S-NIAH Results

Section 5.3 of the main paper reports S-NIAH-1 and S-NIAH-2 results up to 16K for the four methods shown in Figure 4. Here we provide the corresponding numerical results, including two additional baselines not covered in the main figure: Mamba2 [13] and E2E-TTT [53].

E2E-TTT [53] is a recent chunk-wise TTT method that adopts a different inner-loop objective from the key–value binding formulation used in our work and in LaCT [66]. The original paper reports results only within the training context length and provides a JAX implementation. We re-implemented E2E-TTT in PyTorch following the architecture and update rule of the original paper, and trained it under our identical data, tokenizer, and optimization setup. As a sanity check, our re-implementation reaches ceiling accuracy with a sequence length of 512, consistent with other models. E^2-TTT retains a substantial margin over all baselines at every extrapolation length on both S-NIAH variants. At 16K on S-NIAH-1, E^2-TTT$_{\text{SwiGLU}}$ reaches 93.6% while all baselines fall below 26%. On S-NIAH-2,

Table 14: Zero-shot performance on the RULER S-NIAH benchmark suite for 1.3B models. Training context is 2K tokens; columns to the right of 2K are extrapolation. E^2-TTT_{SwiGLU} retains a substantial margin over all baselines at every extrapolation length. Best result per column in bold.

Method	S-NIAH-1						S-NIAH-2					
	512	1K	2K	4K	8K	16K	512	1K	2K	4K	8K	16K
Mamba2	**100.0**	**100.0**	98.6	63.8	31.6	13.0	**100.0**	**99.8**	**99.4**	45.2	27.4	5.0
HQLT	**100.0**	90.0	72.6	50.2	37.4	25.2	**100.0**	80.0	57.0	37.2	15.4	4.4
LaCT	**100.0**	99.4	**100.0**	81.0	37.4	3.0	**100.0**	96.8	91.4	80.6	5.6	0.0
E2E-TTT	**100.0**	60.0	27.6	12.6	6.4	3.0	**100.0**	68.4	29.8	17.4	10.2	3.4
E^2-TTT_{SwiGLU}	**100.0**	96.8	95.6	**95.0**	**95.4**	**93.6**	**100.0**	99.0	96.8	**87.6**	**40.6**	**6.0**
E^2-TTT_{MLP}	**100.0**	95.6	93.2	89.2	88.2	85.0	**100.0**	94.2	79.6	32.6	9.8	4.0

E^2-TTT_{SwiGLU} remains the leading method through 16K, whereas all sub-quadratic baselines drop below 30% from 8K onward.

B.7 Training at a Larger Budget

To verify training feasibility at a larger budget, we additionally trained E^2-TTT_{SwiGLU} at 1.3B parameters for 100B tokens ($6.7\times$ the budget of every other run in this paper) at a 4K training context, leaving the fast-weight network, the sliding-window size (512), the chunk size (512), the data source and the tokenizer unchanged. Table 15 reports the resulting checkpoint. Perplexity and commonsense accuracy improve substantially over the 15B-token model, and in-context retrieval rises from 43.6% to 58.1%. This run serves only as a feasibility check. Since the two rows differ in both the token budget and the training context, the gains cannot be attributed to the larger budget alone. It does, however, show that E^2-TTT trains stably at a budget $6.7\times$ larger than that of the main experiments.

Table 15: A 1.3B E^2-TTT_{SwiGLU} checkpoint trained for 100B tokens at a 4K training context, shown next to the 15B-token checkpoint used throughout the main experiments. The Commonsense and Retrieval columns average the task sets of Tables 2 and 3; retrieval inputs are truncated to 2K and therefore fit inside the training context of both rows.

Training tokens	Train ctx.	Perplexity ↓ Wiki.	Perplexity ↓ LMB.	Commonsense Avg. ↑	Retrieval Avg. ↑
15B	2K	19.9	15.8	53.6	43.6
100B	4K	16.0	10.0	60.1	58.1

Impact Statement and Limitations

E^2-TTT advances the efficiency of long-context sequence modeling by parallelizing token-wise Test-Time Training updates. As a foundational architecture for general-purpose language and multimodal models, it inherits the broad set of societal considerations associated with such models, but does not introduce risks specific to the proposed mechanism beyond those already present in existing TTT and linear-attention variants.

We highlight two limitations of the current study. First, the chunk-wise output step (Eq. 6) applies a fixed fast-weight state to all tokens within a chunk, producing a within-chunk "blind spot" for fine-grained causal dependencies. We mitigate this in practice through the sliding-window attention branch, and developing an efficient token-wise output step remains an interesting direction for future work. Second, following the standard evaluation protocol of recent TTT and linear-attention work [52, 28], our experiments are conducted at scales up to 1.3B parameters. Extending E^2-TTT to larger model sizes is a natural direction for future work.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract's claims are all supported: (i) exactness of the closed-form chunk-end states under the chunk-start gradient approximation (Proposition 3.1); (ii) on par with TTT/hybrid baselines in language modeling and better on in-context retrieval (Sec. 5.1, 5.2); (iii) >90% on passkey retrieval (S-NIAH-1) at 8× training context (Sec. 5.3); (iv) matching the throughput of efficient chunk-wise methods (Appendix B.5).

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The "Impact Statement and Limitations" section explicitly discusses (a) the within-chunk "blind spot" caused by the chunk-wise output step, mitigated by the SWA branch, and (b) the 1.3B-parameter scale ceiling of the empirical study.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

   Justification: Proposition 3.1 is stated in Sec. 3.2 with explicit assumptions. The full derivation is provided in Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Sec. 4 fully specifies the architecture. Appendix B.1 lists all hyperparameters. Appendix B.2 details every evaluation protocol. Additionally, our code and model checkpoints are publicly available at `https://github.com/zeyun-zhong/E2-TTT`.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Our code, training/evaluation scripts, and model checkpoints are publicly available at https://github.com/zeyun-zhong/E2-TTT. All training and evaluation data are publicly available datasets (Appendix B.1, B.2).

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Training and evaluation details are provided in Sec. 5 and Appendix B.1 and B.2.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Error bars are not reported due to the high computational expense involved for all settings.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide the machine specifications and the compute time in Appendix B.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our work complies with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

Justification: The Impact Statement notes that E^2-TTT inherits the broad societal considerations of general-purpose language and multimodal models without introducing risks specific to the proposed mechanism beyond those of existing TTT and linear-attention variants.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: We release only small-scale ($\leq$1.3B) research checkpoints pretrained on the public, filtered FineWeb-Edu corpus, and no new datasets. Given their limited scale and capability, they do not pose a high risk for misuse.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We use publicly available assets (FineWeb-Edu, LLaVA-Video-178K, Qwen3VL, fla-hub tokenizer, and evaluation suites including LongBench, RULER, and lm-evaluation-harness), credit them via citations, and use them in accordance with their respective licenses and terms of use.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The new assets are our code and pretrained model checkpoints, released at `https://github.com/zeyun-zhong/E2-TTT` with documentation on installation, training, evaluation, and checkpoint usage. Model and training details are described in Sec. 4 and Appendix B.1.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: We don't do crowdsourcing and research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: Our paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: The core methodological contribution (the closed-form parallel kernel and the language models trained from scratch) does not involve any external LLM. In Sec. 5.5, we use Qwen3VL-2B-Instruct as a frozen pretrained backbone for a parameter-efficient adaptation experiment; only the inserted E^2-TTT branch is trained. The model is used as published, without prompting, distillation, or data generation. No LLM was used as a methodological component beyond this standard transfer-learning setup.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.