



ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

MASTER PROJECT

COMPARING TEST-TIME TRAINING METHODS FOR LONG-CONTEXT LANGUAGE MODELING

Mikuláš Vanoušek

Supervisors: Nicolas Flammarion, Stanislav Fort, and Jiří Balhar

18th August 2026

ABSTRACT

Test-Time Training (TTT) has recently been proposed as a way to extend sliding-window Transformers to long contexts. Two prominent instantiations use different mechanisms and target different deployment regimes: End-to-End TTT meta-learns a from-scratch initialization with second-order gradients, whereas In-Place TTT uses a closed-form drop-in update for pre-trained LLMs. We present the first controlled head-to-head comparison at 200M-class scale, pre-training models from scratch on FineWeb-Edu at context lengths up to 8K under a shared data, token-budget, and evaluation protocol. To make the comparison methodologically aligned, we test IP-TTT in its secondary SWA/from-scratch regime rather than its primary drop-in setting.

Both TTT methods outperform the sliding-window baseline at longer contexts and recover part of its loss gap to full attention, but neither closes the gap; at 8K, IP-TTT has a 0.003 lower point-estimate loss than TTT-E2E at the reported precision. Because the sweep contains one training seed and the methods use different architectures, this difference does not establish a general ranking. Disabling the test-time update makes both architectures worse than the sliding-window baseline at 4K and 8K, showing that the longer-context gains come from active adaptation rather than a stronger static substrate. On RULER, aggregate scores remain similar, but IP-TTT retrieves 1.58% of needles whose values lie outside the final 1K window, whereas SWA and TTT-E2E retrieve none. When trained directly on synthetic associative recall, IP-TTT recovers 61.3% of out-of-window associations, whereas the tested TTT-E2E configuration recovers less than 0.01%, revealing sharply different memory behavior. Finally, under a 200M-token continuation budget, neither method meaningfully improves a pretrained sliding-window checkpoint, and TTT-E2E exhibits severe transient gradient instability. These results show that test-time adaptation can carry useful information beyond a local attention window, but neither implementation is a complete replacement for full attention or a universal winner across language modeling, retrieval, and deployment.

ACKNOWLEDGMENTS

I thank Nicolas Flammarion for supervising my thesis, Stanislav Fort and Aisle for their generous support and the resources they provided, and Jiří Balhar for his guidance and feedback.

CONTENTS

Acknowledgments	iii
List of Abbreviations	v
1 Introduction	1
2 Related Work	3
3 Methods	6
3.1 Baseline Architectures	6
3.2 Shared TTT Setup	6
3.3 End-to-End TTT	9
3.4 In-Place TTT	10
4 Experiments	13
4.1 200M Context-Length Pre-Training Sweep	13
4.2 RULER Retrieval Diagnostic	17
4.3 Synthetic Key–Value Recall	18
4.4 Retrofitting TTT to a Pretrained SWA Checkpoint	20
5 Conclusion	23
A Additional experiments	28
A.1 Attempted Reproduction of TTT-E2E Context-Extension Scaling	28
A.1.1 Results and Interpretation	29
A.2 Extension-Token Budget Ablation	30
B Implementation	32
C Use of AI tools	34

LIST OF ABBREVIATIONS

BPB	Bits per byte
FA	Full attention
GPU	Graphics processing unit
IP-TTT	In-Place Test-Time Training
K	1024; binary shorthand for token counts (e.g., 8K = 8192 tokens)
KVB	Key-value binding
LLM	Large language model
LM	Language model
MLP	Multilayer perceptron
MQAR	Multi-Query Associative Recall
RMSNorm	Root mean square layer normalization
RoPE	Rotary position embedding
SWA	Sliding-window attention
SwiGLU	Swish-gated linear unit
TTT	Test-Time Training
TTT-E2E	End-to-End Test-Time Training

CHAPTER 1

INTRODUCTION

Self-attention’s $O(T^2)$ prefill cost is a central computational bottleneck for long-context language modeling (Vaswani et al., 2017). Two well-studied strategies trade some of attention’s behavior for sub-quadratic cost: replacing attention with a constant-state-size recurrent layer (Mamba 2 (Dao and Gu, 2024); Gated DeltaNet (Yang, Kautz and Hatamizadeh, 2025)), or restricting it via a sliding window. A third, more recent strategy is Test-Time Training (TTT) (Sun et al., 2025): update a small set of *fast weights* at inference time via a self-supervised objective, providing an explicit adaptive channel for retaining information as the sequence is processed. In this thesis, we study TTT combined with sliding-window attention (SWA), where this channel can carry information after it leaves the local attention window.

Two recent TTT papers for long-context language modeling instantiate this idea differently. End-to-End TTT (TTT-E2E) (Tandon et al., 2025) gives the model an additional set of fast weights that act as a temporary memory. After processing each chunk, the model uses its next-token prediction errors to update these weights, allowing later chunks to benefit from information seen earlier. During pre-training, the model learns a starting point from which these test-time updates are useful.

In-Place TTT (IP-TTT) (Feng et al., 2026) instead turns part of an existing MLP into the model’s temporary memory. After each chunk, it writes token-derived information into the MLP’s down-projection using a direct closed-form update, without backpropagating the prediction loss through the entire model. The two methods therefore implement the same broad idea—letting the model adapt while reading—but differ in what they update, how they update it, and how they are trained.

The two papers evaluate different deployment regimes using disjoint models, datasets, and baselines. TTT-E2E focuses on from-scratch SWA training, whereas IP-TTT primarily targets retrofitting pretrained models and includes secondary from-scratch experiments. Their reported results therefore do not provide a controlled head-to-head comparison. We close this gap with a controlled comparison at 200M-class scale on the FineWeb-Edu dataset (Penedo et al., 2024), with full-attention (FA) and sliding-window-attention Transformers as the two non-TTT baselines. Following the headline plots of both target papers (TTT-E2E Fig. 1; In-Place TTT Fig. 2), we focus on a single axis: held-out next-token loss as a function of pre-training context length, training models from scratch for 4B tokens at context lengths from 1K to 8K while fixing the SWA window at 1K.

A methodological caveat is that this comparison is not the primary deployment setting emphasized by Feng et al. (2026). Their main use case is IP-TTT as a drop-in enhancement for regularly pre-trained LLMs, principally on full-attention backbones, while their SWA/from-scratch experiments are secondary. We nevertheless instantiate IP-TTT with SWA and from-scratch pre-training because this is the closest methodological match to TTT-E2E and isolates the TTT mechanism under a shared long-context training recipe.

RESEARCH QUESTIONS. This setup lets us ask four questions explicitly. First, in the closest controlled regime we can instantiate — shared data, token budget, context lengths, SWA substrate, and from-scratch long-context training — which TTT implementation achieves lower long-context language-modeling loss? Second, how much of each method’s performance comes from its static substrate, and how much from the active test-time update? Third, to what extent can each TTT method support explicit retrieval beyond its attention window? Fourth, can either method improve a pretrained SWA checkpoint under a small continuation budget, and can TTT-E2E learn an effective update without from-scratch meta-training?

CONTRIBUTIONS. We make four contributions. First, to our knowledge, we provide the first head-to-head comparison of TTT-E2E (Tandon et al., 2025) and In-Place TTT (Feng et al., 2026) under a shared data, token-budget, context-length, and evaluation protocol, with full-attention and sliding-window-attention Transformers as non-TTT baselines. Both methods outperform SWA at longer contexts and recover part of its loss gap to FA, but neither closes the gap; at 8K, IP-TTT has a 0.003 lower point-estimate loss than TTT-E2E at the reported precision, which does not establish a general ranking given the single training seed and architectural differences. Second, we apply a *substrate–update decomposition* to both TTT methods: by evaluating each TTT checkpoint with the inner-loop update enabled (TTT) and disabled (TTT-OFF) at test time, we compare the static substrate cost and the benefit of the update under a shared protocol. Third, we test whether long-context language-modeling gains translate into explicit retrieval. On RULER, aggregate scores are similar, but IP-TTT retrieves 1.58% of out-of-window needles while SWA and TTT-E2E retrieve none. On synthetic associative recall, IP-TTT recovers 61.3% of out-of-window associations and the tested TTT-E2E configuration recovers less than 0.01%. These results reveal sharply different retrieval behavior and show that lower long-context loss does not necessarily imply explicit long-range recall. Fourth, we test both methods as small-budget retrofits of a pretrained SWA checkpoint. Under a 200M-token continuation budget, neither method meaningfully improves the checkpoint, and TTT-E2E exhibits severe transient gradient instability.

THESIS STRUCTURE. Chapter 2 positions this comparison within prior work on long-context modeling, test-time training, and retrieval diagnostics. Chapter 3 describes the shared baselines and the architectures and update mechanisms of TTT-E2E and IP-TTT. Chapter 4 presents the controlled language-modeling comparison, retrieval experiments, and pretrained-checkpoint retrofits. The final chapter summarizes the answers to the research questions, limitations, and directions for future work.

CHAPTER 2

RELATED WORK

This chapter reviews the prior work needed to understand the comparison in this thesis and to see which gap it fills. We first review the long-context problem and the established sub-quadratic alternatives to full attention; together they supply our two baselines (FA and SWA) and the substrate on which both TTT methods are built. We then trace the test-time-training line of work from delta-rule fast-weight models to the general TTT framework, introducing the shared vocabulary—fast weights updated by an inner-loop rule—in which the two compared methods are easiest to contrast. Next, we summarize TTT-E2E and In-Place TTT as reported by their original papers, since the claims made there are exactly what our controlled comparison re-examines. Finally, we review the retrieval diagnostics used in our experiments and position this thesis relative to all of the above.

LONG-CONTEXT LANGUAGE MODELING. The methods compared in this thesis exist because full attention is expensive at long context; this paragraph reviews that cost and the two standard sub-quadratic responses: sliding-window attention, which serves as the substrate and baseline for both TTT methods, and recurrent models, which both TTT papers compare against. Self-attention (Vaswani et al., 2017) gives each query direct access to all preceding tokens but pays $O(T^2)$ prefill cost. Sliding-window attention (SWA) (Beltagy, Peters and Cohan, 2020) restricts each query in a layer to a fixed window of k tokens, recovering $O(T)$ prefill while removing direct access to more distant tokens. In practice, SWA is often deployed as part of a hybrid architecture rather than on its own; most notably, Gemma 2 alternates local and global attention layers, while Gemma 3 uses five local layers for every global layer (G. Team, Riviere et al., 2024; G. Team, Kamath et al., 2025). Mamba 2 (Dao and Gu, 2024) instead provides a recurrent alternative with a fixed-size state: each recurrent step costs $O(1)$ in sequence length and full prefill costs $O(T)$, but the model must compress the preceding context into that fixed-size state rather than retaining direct access to individual tokens.

KEY-VALUE-BINDING TEST-TIME TRAINING. Both compared methods descend from an older idea: instead of enlarging the attention window, let part of the model’s weights change while the sequence is being read, so that those weights act as a memory. The simplest instantiation, which also supplies the mechanism IP-TTT builds on, is the delta rule. Delta-rule fast-weight models can be viewed as a form of key-value-binding (KVB) test-time training (TTT): a recurrent matrix is updated online (as we process the sequence of tokens) to map observed keys to values. For a key $\mathbf{k}_t$, value $\mathbf{v}_t$, and linear memory $\mathbf{W}_{t-1}$, the reconstruction objective and one gradient step are

$$\mathcal{L}_t(\mathbf{W}) = \frac{1}{2} \|\mathbf{W}\mathbf{k}_t - \mathbf{v}_t\|_2^2, \quad \mathbf{W}_t = \mathbf{W}_{t-1} + \eta_t (\mathbf{v}_t - \mathbf{W}_{t-1}\mathbf{k}_t) \mathbf{k}_t^\top. \quad (2.1)$$

The difference $\mathbf{v}_t - \mathbf{W}_{t-1}\mathbf{k}_t$ is the current retrieval error: what the memory should return for $\mathbf{k}_t$ minus what it returns before the update. The update changes the memory to reduce this error, so applying it later to the same or a similar key produces an output closer to $\mathbf{v}_t$; a general query $\mathbf{q}$ retrieves $\mathbf{W}_t\mathbf{q}$. Gated DeltaNet (Yang, Kautz and Hatamizadeh, 2025) combines this delta-rule update with data-dependent gating, while Kimi Delta Attention extends it with finer-grained gating (K. Team et al., 2025).¹

The delta rule is one instance of a broader framework. TTT in general treats sequence processing as online learning: fast weights are updated at inference time using a self-supervised objective. Sun et al. (2025) instantiate this framework with per-token KVB reconstruction updates as a drop-in self-attention replacement; T. Zhang et al. (2025) swap per-token for large-chunk updates for higher hardware utilization.

END-TO-END TTT (TANDON ET AL., 2025). The first of the two methods we compare departs from KVB-style TTT in what the inner loop optimizes. TTT-E2E makes both training and test loss *end-to-end*: the layer-wise KVB reconstruction loss is replaced by the next-token-prediction loss applied at the end of the network, and the initialization W_0 is meta-learned by an outer loop that minimizes the loss *after* the inner-loop TTT steps, requiring gradients of gradients during pre-training. The attention substrate is SWA (window $k=8K$) in every block, with mini-batch ($b=1K$) fast-weight updates to the MLPs of the last 25% of layers; in each updated block, the fast MLP is followed by a static second MLP that preserves pre-trained knowledge. The FlashAttention-3 kernel (Shah et al., 2024) does not support the second-order derivatives required by this meta-training objective, so TTT-E2E pre-training cannot use it. In their long-context scaling experiments, Tandon et al. (2025) report that the loss curves of Mamba 2 and Gated DeltaNet flatten or worsen beyond a few tens of thousands of tokens, unlike full attention. At 3B parameters and 164B tokens, TTT-E2E maintains a constant loss gap to full attention up to 128K context, while SWA, Mamba 2, and Gated DeltaNet do not (Figure 2.1). The method is presented as a from-scratch training recipe, and its dependence on a jointly meta-learned initialization makes retrofitting nontrivial.

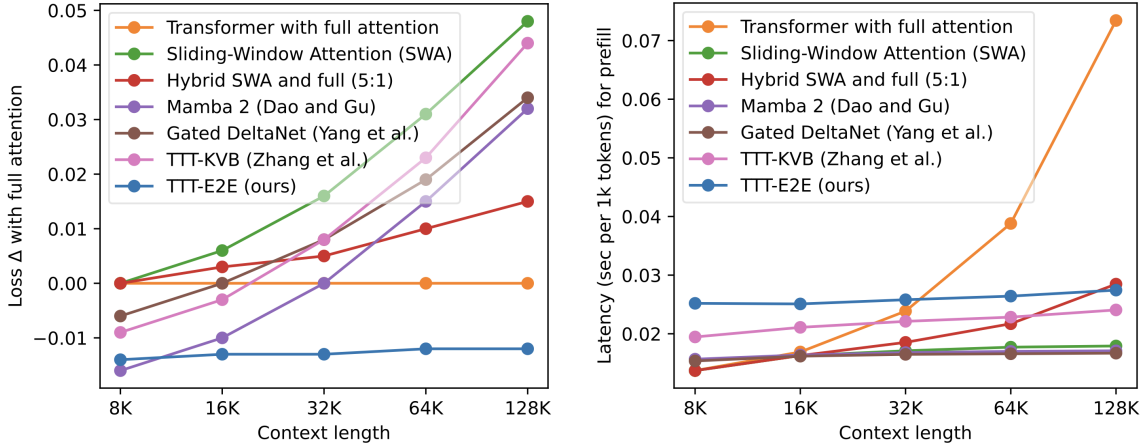


FIGURE 2.1

Figure 1 from Tandon et al. (2025). Loss (relative to the FA baseline) and latency as a function of context length. It shows the reported advantage of TTT-E2E: it maintains performance superior to FA while achieving constant per-token inference latency with respect to context length, like SWA.

IN-PLACE TTT (FENG ET AL., 2026). Whereas TTT-E2E is a from-scratch training recipe with a meta-learned initialization, IP-TTT, the second compared method, is designed primarily as a *drop-in*

¹Kimi Delta Attention is used as part of the hybrid attention architecture in Kimi K3, released in July 2026 (Kimi K3 Tech Blog: Open Frontier Intelligence 2026).

enhancement: it introduces no new fast-weight matrix in the MLP path, repurposing the gated-MLP down-projection $\mathbf{W}_d$ as fast weights. For each chunk of C tokens, it updates this matrix in closed form using the token activations and an LM-aligned Conv1D target that mixes nearby token embeddings, including future positions.² Under an induction-head abstraction, this target provably raises the correct-token logit in expectation while a generic reconstruction target does not. Updates are chunk-wise ($C=512\text{--}1024$) with no inner-loop autodiff. The paper’s main experiments apply it as a drop-in continual-pre-training step ($\sim 20\text{--}35\text{B}$ tokens) on top of Qwen3-4B / LLaMA-3.1-8B / Qwen3-14B, improving RULER scores at every length up to 128K with extrapolation to 256K; its secondary from-scratch experiments at 500M and 1.5B use an SWA backbone and beat SWA, Gated Linear Attention, DeltaNet, and Large-Chunk Test-Time Training in sliding-window perplexity.

LONG-CONTEXT RETRIEVAL DIAGNOSTICS. Our experiments do not stop at language-modeling loss: Sections 4.2 and 4.3 test whether TTT’s loss improvements correspond to explicit retrieval, using two diagnostics we review here. Language-modeling loss averages predictive performance over tokens and does not directly measure whether a model can retrieve a particular fact from distant context. RULER (Hsieh et al., 2024) evaluates explicit long-context capabilities through synthetic tasks embedded in natural-language prompts, including needle-in-a-haystack retrieval. Multi-Query Associative Recall (MQAR) (Arora et al., 2023) more directly isolates the ability to bind keys to values and recall those associations when a key reappears.

POSITIONING. Prior work therefore establishes TTT-E2E’s from-scratch long-context scaling and IP-TTT’s gains in its primary pretrained-model regime, but differences in models, data, baselines, and deployment setting prevent a controlled comparison. This thesis compares the methods in their closest shared from-scratch SWA regime and separately tests whether their language-modeling improvements correspond to explicit long-range retrieval.

²This does not leak future information into predictions: the target for a chunk is used only after that chunk has been processed, and the resulting fast weights affect only subsequent chunks; see Section 3.4.

CHAPTER 3

METHODS

This chapter first defines the shared full-attention and sliding-window-attention Transformer architectures, then presents In-Place Test-Time Training (IP-TTT) (Feng et al., 2026) and End-to-End Test-Time Training (TTT-E2E) (Tandon et al., 2025) as modifications of the sliding-window baseline.

3.1 BASELINE ARCHITECTURES

The full-attention (FA) and sliding-window-attention (SWA) baselines use the same decoder-only Transformer architecture and differ only in their attention masks. Each Transformer block applies an attention sublayer followed by a SwiGLU MLP sublayer (Shazeer, 2020). Both sublayers use RMSNorm (B. Zhang and Sennrich, 2019) before and after the transformation, followed by a residual addition.

Let $\mathbf{H} \in \mathbb{R}^{n \times d}$ be the residual-stream hidden states entering the MLP sublayer after the attention residual connection, with one row per token position and d hidden features. The actual input to the MLP is the normalized matrix $\mathbf{X} = \mathcal{N}_{\text{pre}}(\mathbf{H})$. The gate and up projections have shape $\mathbf{W}_g^{(l)}, \mathbf{W}_u^{(l)} \in \mathbb{R}^{d_{\text{mlp}} \times d}$, and the down projection has shape $\mathbf{W}_d^{(l)} \in \mathbb{R}^{d \times d_{\text{mlp}}}$. The ordinary MLP and its residual update in block l are

$$\begin{aligned} \mathbf{X} &= \mathcal{N}_{\text{pre}}(\mathbf{H}), \\ \text{MLP}_l(\mathbf{X}) &= \left[\phi(\mathbf{X}\mathbf{W}_g^{(l)\top}) \odot (\mathbf{X}\mathbf{W}_u^{(l)\top}) \right] \mathbf{W}_d^{(l)\top}, \\ \mathbf{H}_{\text{out}} &= \mathbf{H} + \mathcal{N}_{\text{post}}(\text{MLP}_l(\mathbf{X})). \end{aligned} \tag{3.1}$$

Here ϕ is the SiLU (or Swish) activation, $\odot$ is elementwise multiplication, and the gated product implements SwiGLU. The operators $\mathcal{N}_{\text{pre}}$ and $\mathcal{N}_{\text{post}}$ are the RMSNorm layers before and after the MLP.

FA allows every token to attend directly to its complete causal prefix (Vaswani et al., 2017). SWA instead restricts direct attention to the most recent k positions permitted by the causal mask (Beltagy, Peters and Cohan, 2020). Stacking SWA blocks gives a wider theoretical receptive field through indirect propagation across layers, but unlike FA it does not give every layer direct access to the complete prefix. The exact model dimensions, attention window, and parameter counts used in the experiments are specified in Chapter 4.

3.2 SHARED TTT SETUP

Both methods augment local attention with an adaptive fast-weight state intended to preserve information after tokens leave the attention window. Sliding-window attention keeps the inference cost of long-context

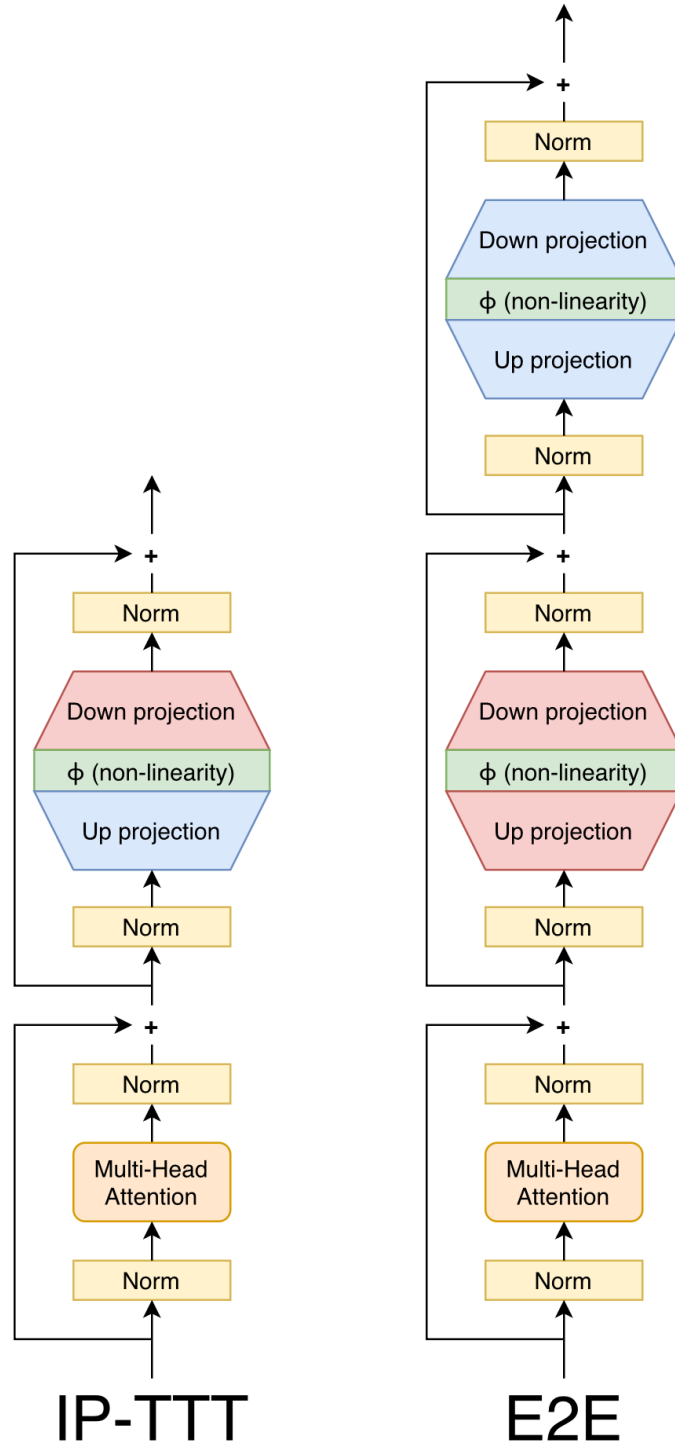
processing manageable, but in each layer a token can attend directly only to the previous k tokens. Test-time training adds another channel for carrying information forward: some of the model weights are allowed to change while the sequence is processed. In the matched comparison, both TTT methods build on the SWA baseline. SWA is therefore the local-attention baseline that TTT is intended to improve, while FA serves as the global-context reference.

The sequence is not processed with one fixed set of weights. Instead, it is split into chunks of C tokens. Predictions inside chunk i use the weights available at the start of that chunk. After the chunk is processed, the model takes a gradient step to update a selected subset of weights, and these updated weights are used for the next chunk. We call the updating parameters *fast weights*; the remaining parameters are *slow weights* and stay fixed during inference. We refer to the between-chunk gradient step as the *inner-loop update*. This distinguishes it from the *outer-loop update*: the training-time gradient step taken after the full sequence loss has been evaluated.

This makes both methods instances of meta-learning. The model is trained to work well with a rule that will change part of its state at test time. Because the updated fast weights persist between chunks, they provide an explicit channel for carrying information beyond the local attention window, alongside the indirect propagation available through stacked SWA layers.

To limit the cost of this additional state, both methods update only part of the MLP and only in selected layers. At inference, only the current fast-weight state must be retained for each active sequence; states from earlier chunks can be discarded, but this still adds per-sequence state beyond the ordinary model weights. During training, differentiating the outer loss through the update trajectory requires retaining or recomputing intermediate states, creating a larger memory overhead than at inference.

COMPARISON AXES. The two methods differ in four main ways. First, they update different weights. TTT-E2E uses an entire prime MLP as dedicated fast-weight capacity and follows it with a static ordinary MLP, while IP-TTT reuses existing MLP down-projection matrices. Second, they use different update objectives. TTT-E2E uses the model’s final next-token prediction loss, while IP-TTT uses a local MLP-level target derived from nearby token embeddings. Third, they implement the update differently. TTT-E2E uses a true back-propagated gradient step through the Transformer, while IP-TTT manually inlines the corresponding inner-loop update as a closed-form sum of outer products. Fourth, they are designed for different deployment regimes. TTT-E2E is a from-scratch training recipe, while IP-TTT is primarily designed as a drop-in enhancement for already pretrained models.

**FIGURE 3.1**

A single TTT-equipped layer in IP-TTT vs TTT-E2E. Fast weights are highlighted in red. The IP-TTT layer is identical to the SWA baseline Transformer, except that it dedicates the MLP down projection to the fast weights. A TTT-E2E layer instead applies a fast prime MLP and a static ordinary MLP sequentially, with a separate residual connection around each MLP.

3.3 END-TO-END TTT

FAST WEIGHTS. TTT-E2E applies fast weights to a selected suffix of the Transformer. The original paper recommends the last quarter of the blocks based on its ablations (Tandon et al., 2025); the exact placements used in our experiments are specified in Chapter 4. In these TTT-enabled layers, the fast weights form a prime MLP that is followed by the static ordinary MLP from Eq. 3.1, as seen in Figure 3.1. Both are residual sublayers: the prime MLP first updates the residual stream, and the ordinary MLP then receives this updated stream as its input. The prime MLP has the same SwiGLU form and projection shapes as the baseline MLP, but uses separate primed matrices. Its computation and the two sequential residual updates are

$$\begin{aligned} \mathbf{X}' &= \mathcal{N}'_{\text{pre}}(\mathbf{H}), \\ \text{MLP}'_{l,i}(\mathbf{X}') &= \left[\phi(\mathbf{X}' \mathbf{W}_{g'}^{(l,i)\top}) \odot (\mathbf{X}' \mathbf{W}_{u'}^{(l,i)\top}) \right] \mathbf{W}_{d'}^{(l,i)\top}, \\ \tilde{\mathbf{H}} &= \mathbf{H} + \mathcal{N}'_{\text{post}}(\text{MLP}'_{l,i}(\mathbf{X}')), \\ \tilde{\mathbf{X}} &= \mathcal{N}_{\text{pre}}(\tilde{\mathbf{H}}), \\ \mathbf{H}_{\text{out}} &= \tilde{\mathbf{H}} + \mathcal{N}_{\text{post}}(\text{MLP}_l(\tilde{\mathbf{X}})). \end{aligned} \quad (3.2)$$

The primed normalization operators belong to the prime MLP, while the unprimed operators belong to the ordinary MLP defined in the baseline architecture; all four normalization layers remain slow. The red primed matrices are the fast weights for chunk i : they form the prime MLP and are updated between chunks, while the ordinary MLP weights remain slow. This gives the method dedicated adaptive capacity: the part of the model that changes across chunks is separated from the ordinary MLP.¹

UPDATE OBJECTIVE. TTT-E2E uses the standard next-token prediction loss for the inner-loop update. Let $\mathbf{W}_i$ denote all prime-MLP fast weights at the start of chunk i , and let $\mathcal{L}_i(\mathbf{W}_i)$ be the mean next-token negative log-likelihood over predictions in that chunk. After making all predictions in chunk i , TTT-E2E applies one gradient step,

$$\mathbf{W}_{i+1} = \mathbf{W}_i - \eta_{\text{E2E}} \nabla_{\mathbf{W}_i} \mathcal{L}_i(\mathbf{W}_i). \quad (3.3)$$

The resulting weights affect only later chunks, so the update cannot change predictions within the chunk that produced it. At the beginning of each sequence, the fast weights are reset to the checkpoint’s learned initialization $\mathbf{W}_0$ and are never carried across documents. The main advantage is that the update is directly aligned with the objective we care about. The same loss that trains the language model is also used to decide how the fast weights should change at test time.²

UPDATE IMPLEMENTATION. Since the loss value can only be computed from the end of the network, the inner-loop update is implemented as an autodiff-computed gradient step. To update the fast weights, we have to backpropagate through the Transformer. During training, the outer loop also differentiates through this inner-loop update. This requires gradients of gradients and makes the method second-order from the auto-differentiation engine’s perspective. Because the FlashAttention-3 kernel does not support the required second-order derivatives, TTT-E2E pre-training cannot use it. Figure 3.2 contrasts this chunked, meta-learned training structure with ordinary pre-training.

¹This extra MLP would significantly increase the parameter count if we left the rest of the model unchanged. As we explain in Section 4.1, we therefore reduce the ordinary MLP width so that the total model size stays almost identical to the FA and SWA baselines.

²It also gives this technique its name. The end-to-end update is computed by backpropagating through the whole Transformer to the fast weights. This is not how previous TTT methods have implemented the inner-loop update.

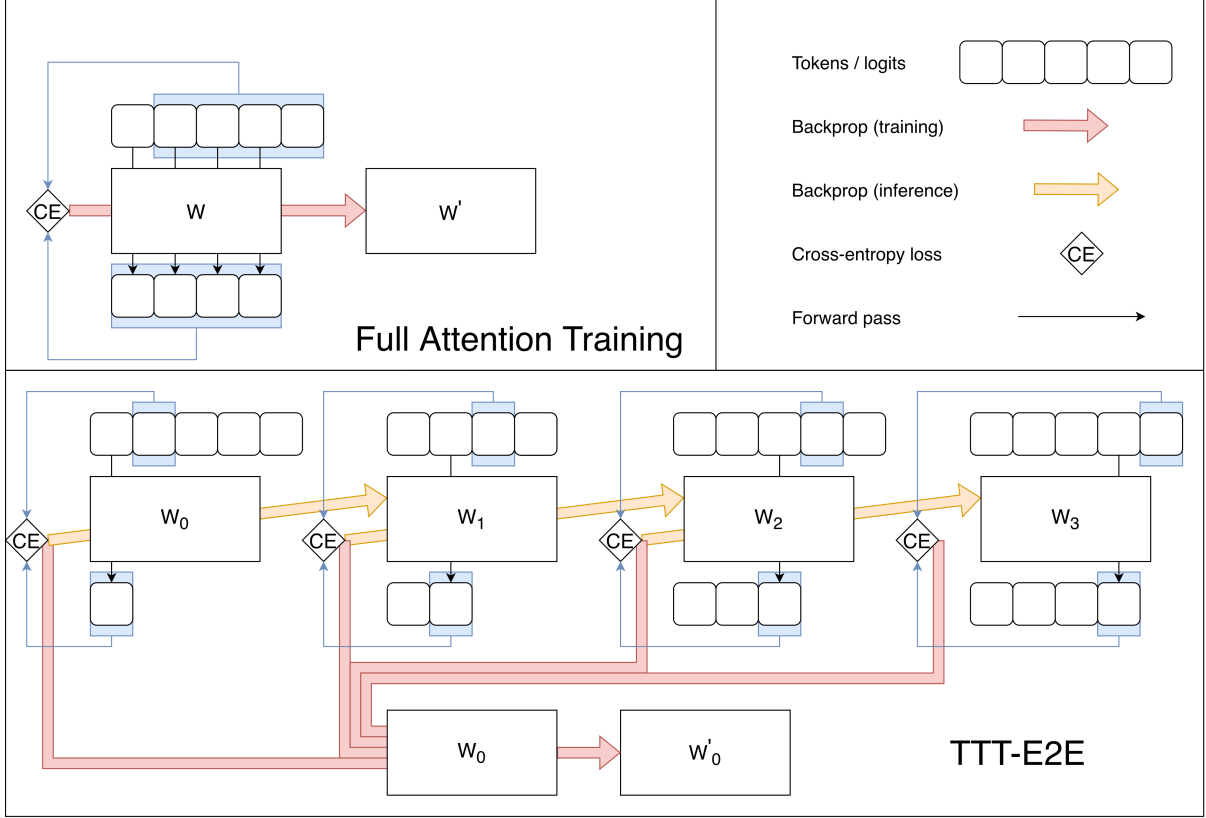


FIGURE 3.2

Schematic of TTT-E2E (Tandon et al., 2025). Top: ordinary full-attention training updates one weight state by back-propagating the language-modeling loss. Bottom: TTT-E2E unrolls the sequence into chunks, applies an inference-time gradient update from $\mathbf{W}_i$ to $\mathbf{W}_{i+1}$ after each chunk, and meta-learns the initial fast weights by back-propagating through the unrolled inner updates during training.

DEPLOYMENT REGIME. TTT-E2E is not a drop-in retrofit. The initialization has to be learned from scratch with the update rule in the training loop. The learned weights are optimized so that future inner-loop updates improve the next-token loss.

3.4 IN-PLACE TTT

FAST WEIGHTS. IP-TTT updates existing MLP down-projection matrices rather than adding a new prime MLP. Concretely, it reuses selected gated-MLP $\mathbf{W}_d / \mathbf{W}_{\text{down}}$ matrices as fast weights. Using the baseline notation, define the gated activations $\mathbf{Z}_l(\mathbf{X})$ and the IP-TTT MLP as

$$\begin{aligned} \mathbf{Z}_l(\mathbf{X}) &:= \phi(\mathbf{X}\mathbf{W}_g^{(l)\top}) \odot (\mathbf{X}\mathbf{W}_u^{(l)\top}), \\ \text{MLP}_{l,i}^{\text{IP}}(\mathbf{X}) &= \mathbf{Z}_l(\mathbf{X}) \mathbf{W}_d^{(l,i)\top}. \end{aligned} \tag{3.4}$$

IP-TTT therefore leaves the gated activations unchanged and replaces only the static down-projection with its chunk-dependent fast-weight state. The red down-projection is the fast weight: it is part of the ordinary MLP computation, but its value changes between chunks. This is what makes the method in-place: the adaptive state shares capacity with the ordinary MLP instead of living in a separate module.

The advantage is that the method can be added with much less architectural change. The disadvantage is that the adaptive state is less isolated than in TTT-E2E. The same matrix has to serve as an ordinary MLP down-projection and as the memory that changes across chunks.

UPDATE OBJECTIVE. IP-TTT does not use the model’s final next-token prediction loss for the inner-loop update. Instead, it creates a local training signal inside the MLP. The goal of this signal is to make the selected MLP down-projection write information that is useful for future next-token predictions. Rather than reconstructing the current-token representation, IP-TTT aims to align the MLP output with the next-token embedding.³ In the implementation, this exact target is relaxed to a learned mixture of nearby token embeddings. For chunk i , IP-TTT constructs

$$\hat{\mathbf{V}}_{[i]} = \text{Conv1D}(\mathbf{X}_{0,[i]}) \mathbf{W}_{\text{target}}, \quad (3.5)$$

where $\mathbf{X}_{0,[i]}$ denotes the input tokens’ embeddings in the i -th chunk, Conv1D is a learned depthwise convolution with kernel size 5, and $\mathbf{W}_{\text{target}}$ is a learned projection back into the model hidden dimension. Both the convolution parameters and the target projection are ordinary slow weights trained by the outer loop, rather than fast weights updated between chunks. We use kernel size 5, following Feng et al. (2026), and do not tune this hyperparameter. The convolution is applied independently within each chunk using symmetric zero padding. At the right boundary, kernel positions that would extend into the next chunk therefore receive zeros rather than future-token embeddings. The target for chunk i is used only to update the down-projection after chunk i has already been processed. Thus any information extracted from tokens in chunk i can affect later chunks, but it cannot change predictions inside the same chunk.

Let $\mathbf{H}_{[i]}$ denote the residual-stream states entering the selected MLP in chunk i , and let $\mathbf{X}_{[i]} = \mathcal{N}_{\text{pre}}(\mathbf{H}_{[i]})$ be the normalized MLP input. Applying Eq. 3.4 to this chunk gives

$$\mathbf{Z}_{[i]} := \mathbf{Z}_l(\mathbf{X}_{[i]}) \in \mathbb{R}^{C \times d_{\text{mlp}}}. \quad (3.6)$$

IP-TTT asks the MLP down-projection output to align with the target using the similarity loss

$$\begin{aligned} \mathcal{L}_{[i]} &= - \left\langle \mathbf{Z}_{[i]} \mathbf{W}_d^{(l,i)\top}, \hat{\mathbf{V}}_{[i]} \right\rangle_F, \\ \nabla_{\mathbf{W}_d^{(l,i)}} \mathcal{L}_{[i]} &= - \hat{\mathbf{V}}_{[i]}^\top \mathbf{Z}_{[i]}, \\ \mathbf{W}_d^{(l,i+1)} &= \mathbf{W}_d^{(l,i)} + \eta \hat{\mathbf{V}}_{[i]}^\top \mathbf{Z}_{[i]}. \end{aligned} \quad (3.7)$$

Here $\langle \mathbf{A}, \mathbf{B} \rangle_F := \sum_{t=1}^C \sum_{j=1}^d A_{tj} B_{tj}$ denotes the Frobenius inner product. The scalar η is the IP-TTT inner-loop learning rate, which scales each fast-weight update. This update stores an association between the gated-MLP activations $\mathbf{Z}_{[i]}$ and the target vectors $\hat{\mathbf{V}}_{[i]}$ in the down-projection. The gradient with respect to the fast weights is a sum of C outer products, equivalently a matrix update of rank at most C , so the inner loop has a closed form. Its main limitation is objective mismatch. The update is aligned with next-token prediction by design, but it is still a local surrogate rather than the model’s actual cross-entropy loss on the final vocabulary logits.

UPDATE IMPLEMENTATION. IP-TTT manually writes the inner-loop gradient update into the model. We do not need the autograd framework to backpropagate through the whole Transformer to construct the inner update. Instead, the update is implemented as ordinary model dynamics: compute the target, compute the gated-MLP activations, and apply the closed-form outer-product update to the selected down-projection matrices. During training, the outer loss still backpropagates through this sequence of analytically specified update dynamics to train the slow weights. Because the inner-loop update never backpropagates through the attention layers, IP-TTT remains compatible with FlashAttention kernels. This compatibility is a practical advantage over TTT-E2E.

³With tied input and output embeddings, the final vocabulary logit for token v is proportional to the inner product between the model’s final hidden state and the same embedding vector used to represent token v at the input. Therefore, moving a hidden state in the direction of the next token’s embedding directly raises that token’s logit relative to embeddings in other directions.

DEPLOYMENT REGIME. Although IP-TTT was designed primarily as a drop-in enhancement for pretrained full-attention LLMs, our matched comparison uses the from-scratch SWA regime justified in the Introduction. This gives both methods the same local-attention substrate and the same opportunity to use fast weights to carry information beyond the attention window.

CHAPTER 4

EXPERIMENTS

This chapter evaluates the methods through four experiments. We first compare held-out language-modeling loss across pre-training context lengths and use update-disabled ablations to separate the contribution of the active update from that of each trained substrate. We then test explicit retrieval using RULER and a synthetic associative-recall task. Finally, we examine whether either method can improve a pretrained SWA checkpoint under a small continuation budget.

4.1 200M CONTEXT-LENGTH PRE-TRAINING SWEEP

QUESTION. The first goal of this thesis is to produce a head-to-head comparison of the two test-time training techniques: End-to-End vs. In-Place. We also include two baselines: full attention and sliding-window attention. How does the next-token loss change when the whole pre-training run is carried out at different context lengths: 1024, 2048, 4096, and 8192 tokens?

The methodology we use for the comparison is considerably different from that of Tandon et al. (2025); for a comparison and justification, see Appendix A.

DATA AND TOKENIZATION. We use FineWeb-Edu (Penedo et al., 2024), a publicly available subset of FineWeb selected for educational content, specifically its `sample-10BT` subset. We chose it because it provides diverse, high-quality pre-training text at a manageable scale and contains enough long documents for training at every context length in our sweep. We derive a 32,000-entry BPE vocabulary from the Llama 3.1 tokenizer (Grattafiori et al., 2024). We retain its base symbols, excluding added and special tokens, and traverse the original merge rules in priority order, retaining valid merges until the target vocabulary size is reached. The resulting tokens receive contiguous IDs. Excluded merges therefore decompose into smaller retained subword units rather than mapping to an unknown token. We preserve the original pre-tokenizer and decoder and do not insert BOS, EOS, or document-boundary tokens.

SEQUENCE CONSTRUCTION AND SPLIT. We filter once for documents containing at least 8K tokens and use the same resulting document population for every run. Sequences are contiguous, non-overlapping chunks from individual documents; documents are never concatenated, and the same 8K source blocks are split into shorter chunks for the shorter-context runs. We then reserve 5% of the filtered documents for evaluation and use the remaining 95% for pre-training. This prevents dataset composition from changing with context length, making differences between runs easier to attribute to the model and context length rather than to the training data. All models are evaluated on exactly the same validation documents.

HELD-OUT EVALUATION. The held-out partition contains 8,873 documents; at 8K context, evaluation uses 13,747 complete sequences, or approximately 112.6M scored next-token predictions. Each evaluation sample contains $T + 1$ consecutive text tokens: the first T are inputs and the following T are targets shifted by one position. We score every target in every complete sequence, including those at sequence edges, and average negative log-likelihood uniformly over these tokens; incomplete trailing chunks are excluded. No special boundary tokens are added, and model state, including fast weights, is reset for every evaluation sequence rather than carried between chunks from the same document.

BASELINES. As the full attention baseline, we train a 200M-class decoder-only Transformer family with 13 layers, hidden size 1024, 16 attention heads, MLP width 2816, tied embeddings, RMSNorm (B. Zhang and Sennrich, 2019), query-key normalization (Henry et al., 2020), RoPE with base 500,000, no dropout, and a 32,000-token vocabulary. The full attention baseline has 199.81M trainable parameters. The sliding window attention baseline has exactly the same shape, but the attention mask is restricted to a fixed 1024 token window.

TTT-E2E & IN-PLACE TTT. Both TTT methods use the same local context scale as the SWA baseline: a 1024-token attention window and a 1024-token TTT update granularity. Thus each method performs one test-time gradient update per 1024 tokens. TTT-E2E uses one inner-loop gradient step per chunk with learning rate 1.0 and gradient clipping at 1.0. Fast weights are reset to the checkpoint’s learned initialization at the beginning of every sequence and are never carried between documents. At a sequence length of 1K, the SWA window covers the complete causal prefix, making its attention mask mathematically identical to FA, and the first TTT update occurs only after all predictions have been made, so it cannot affect the loss. We nevertheless run all four methods at 1K as sanity checks of the matched training and evaluation methodology; the substantive long-context comparison begins at 2K.

For TTT-E2E, we follow the paper’s recipe of applying fast weights only in the final quarter of the stack (Tandon et al., 2025). In our 13-layer model this corresponds to the final three blocks. Since adding prime MLPs would otherwise increase the parameter count substantially, we reduce the ordinary MLP width to 2288 in all layers and use the same width for the prime MLPs in the updated suffix blocks, giving 199.82M parameters, essentially identical to FA/SWA.

For IP-TTT, we follow the paper’s every-sixth-layer recipe (Feng et al., 2026) and enable updates in layers 7 and 13 with inner-loop learning rate $\eta = 3$. IP-TTT leaves the ordinary MLP shape unchanged and reuses the selected down-projections as fast weights. The learned target projection and depthwise Conv1D add slow-weight parameters, bringing the IP-TTT model to 213.51M parameters (+6.9% relative to FA/SWA).

TRAINING. Each run uses a fixed token batch of 524,288 tokens per optimizer step: global batch sizes 512, 256, 128, and 64 for sequence lengths 1K, 2K, 4K, and 8K respectively. We train for 4B total tokens, which corresponds to 20 training tokens per parameter as recommended by Hoffmann et al. (2022). We define one pre-training epoch as one pass over the complete 8K source blocks extracted from the 95% training partition, comprising approximately 2.14B scored tokens. Because the shorter-context runs split these same source blocks, they share this denominator; the 4B-token budget therefore corresponds to 1.87 epochs for every context length.

We use the AdamW optimizer (Loshchilov and Hutter, 2019), with $\beta_1 = 0.9$, $\beta_2 = 0.95$, peak learning rate 2×10^{-3} , cosine decay to 10^{-5} , 10% linear warmup steps, weight decay 0.1, and gradient clipping at 1.0. The reported runs use model and data-shuffling seeds of zero. GPU-hour accounting for the minimal set of runs used in the reported results is summarized in Appendix Table B.2.

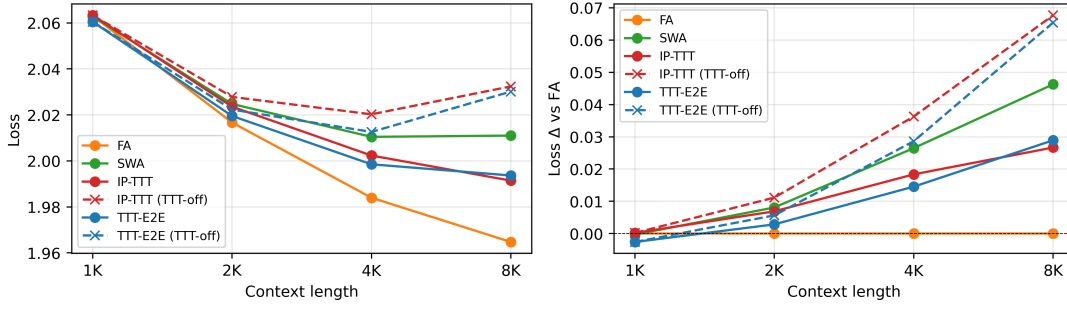


FIGURE 4.1

Held-out loss for 200M-class pre-training runs as a function of pre-training context length. The right panel reports loss relative to the FA baseline at the same context length. At 1K, the SWA mask is identical to FA and the TTT update occurs only after all predictions; these points therefore serve as sanity checks of the matched methodology.

TABLE 4.1

Held-out next-token loss for the 200M-class pre-training sweep, containing the same values shown in Figure 4.1.

Method	1K	2K	4K	8K
FA	2.063	2.017	1.984	1.965
SWA	2.063	2.025	2.010	2.011
IP-TTT	2.063	2.024	2.002	1.991
IP-TTT (TTT-OFF)	2.063	2.028	2.020	2.032
TTT-E2E	2.061	2.020	1.999	1.994
TTT-E2E (TTT-OFF)	2.061	2.022	2.013	2.030

RESULTS. At 1K, FA, SWA, and IP-TTT are mathematically equivalent for prediction: the SWA window covers the complete causal prefix, and the IP-TTT update occurs only after all predictions have been made. All three obtain held-out loss 2.063. TTT-E2E obtains 2.061; although its update likewise cannot affect predictions, its static shape differs because parameter matching narrows the ordinary MLPs and adds parallel prime MLPs in the final three layers. The TTT-OFF values are unchanged at the reported precision, confirming that no test-time update affects predictions at this length.

Figure 4.1 shows that FA benefits most from increasing the pre-training context in this regime. As the sequence length grows, SWA falls behind FA. The point estimates place both TTT methods below SWA at every context length from 2K to 8K. Neither method closes the full gap to FA. At 8K, IP-TTT has loss 1.991 and TTT-E2E has loss 1.994, so the point estimate favors IP-TTT by 0.003 at the reported precision. This comparison is between complete method-specific architectures rather than only their update rules: parameter matching narrows TTT-E2E’s ordinary MLPs in every layer and adds parallel prime MLPs in the final three layers, whereas IP-TTT retains the baseline MLP shape and has 6.9% more total parameters. Together with the single completed training seed per configuration, these differences prevent interpreting the observed gap as a general ranking of the update methods.

The TTT-OFF rows are inference-time ablations of the same checkpoints, which were trained expecting the update. They therefore measure, operationally, how much each trained checkpoint relies on its active update rather than the best performance its architecture could achieve without TTT. Disabling the update produces a different ordering. The TTT-OFF variants become increasingly worse than SWA as context length grows, particularly at 4K and 8K. For these checkpoints, the active updates are therefore necessary both to overcome the deficit observed under update-disabled execution and, at longer contexts, to surpass

SWA and recover part of its loss gap to FA.

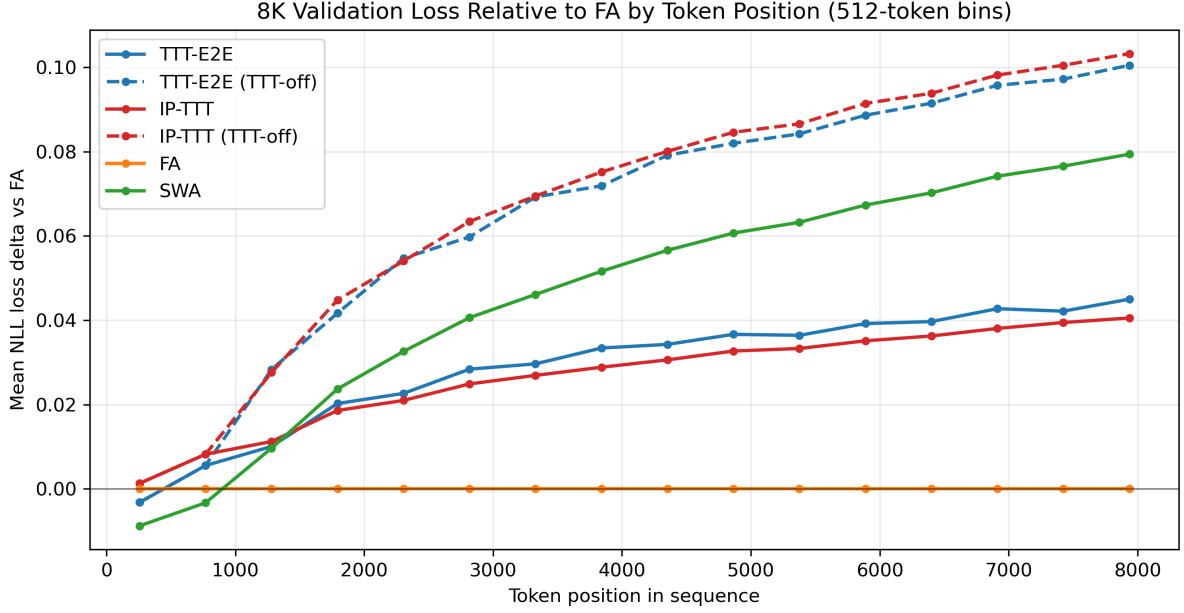


FIGURE 4.2

Held-out next-token loss by position for the completed 8K, 200M-class runs, plotted relative to the FA loss at the same token position. Each point averages a 512-token position bin after first averaging each position over the full held-out split. FA is therefore the zero line by construction; positive values mean the model is worse than FA at that position, while negative values mean it is better.

Figure 4.2 shows where the 8K loss gap appears inside the sequence. All non-FA curves are close to FA near the beginning, where the 1K SWA window covers most or all available history. After the local window saturates, the losses drift upward relative to FA, showing that the gap in Figure 4.1 is not uniform over positions: it is concentrated in later tokens whose useful context can extend beyond the fixed local window. SWA degrades steadily as an increasing share of the prefix lies beyond each layer’s direct attention window. The stacked SWA network can still propagate information indirectly across layers, so its network-level receptive field is larger than 1K; the comparison shows that this indirect route is less effective here than FA’s direct global access. Both active TTT methods remain closer to FA than SWA over later positions, while their TTT-OFF variants become substantially worse than SWA. The benefit of the active update thus appears precisely in the later-position regime where local attention is most limited. Nevertheless, the active TTT curves also drift away from FA with position, showing that neither update recovers full-attention performance.

TAKEAWAY. Both TTT methods recover part of the long-context loss gap, and the TTT-OFF ablation shows that their trained checkpoints depend on the active update to do so. Because these checkpoints were optimized with TTT enabled, however, the ablation does not characterize independently optimized static versions of the architectures. Neither method matches FA. At 8K, IP-TTT’s point estimate is lower by 0.003 than TTT-E2E, but the single training seed and the methods’ different architectures prevent interpreting this gap as a general ranking. At this model and data scale, TTT is therefore a useful but incomplete alternative to global attention. More specifically, our results show substantially more modest performance for TTT-E2E than the original paper: it remains behind FA and performs similarly to IP-TTT rather than exhibiting the strong scaling advantage reported by Tandon et al. (2025). This casts doubt on

how broadly that advantage generalizes across model scales and training budgets.¹

4.2 RULER RETRIEVAL DIAGNOSTIC

QUESTION. Tandon et al. (2025) report that TTT-E2E does not improve retrieval performance on the RULER needle-in-a-haystack task, despite improving language-modeling loss. Feng et al. (2026) report that IP-TTT improves retrieval performance. Do these observations reproduce in our setting?

PROTOCOL. We evaluate the models pre-trained at 8K context in Section 4.1 on the RULER needle-in-a-haystack task (Hsieh et al., 2024), using RULER commit `ab17b785`. We use the easiest task, `niah_single_1`. Each example uses a generated word key and number value; for example, the model may need to recover the value “4612365” associated with a particular key. This needle is placed randomly in a haystack of unrelated, repetitive text (“The grass is green. The sky is blue. ...”). The haystack contains no other number, which makes the needle stand out among the distractors and makes retrieval easier than in the other RULER tasks. We chose this variant because it already challenges the small 200M FA baseline.

We use RULER’s base-model prompt template and answer prefix, without a chat template. Inputs are generated and tokenized with the Llama 3.1 8B tokenizer truncated to the same 32,000-token vocabulary used in pre-training. The maximum total length is 8192 tokens, matching the checkpoints’ pre-training length, with up to 128 tokens reserved for greedy generation. All models are evaluated on the same 5000 generated examples. We report the official RULER task accuracy. After prefilling the prompt, we greedily decode the model’s response; the official `string_match_all` scorer marks an example correct when the ground-truth answer occurs as a case-insensitive substring of the generated response.

For SWA, the attention window is 1K tokens. For TTT-E2E and IP-TTT, the attention window is also restricted to 1K tokens, but they have the opportunity to use their fast weights to store some information about tokens outside this window. We split the examples according to whether the first token of the needle value lies within the final 1024 prompt tokens available when generation begins. This gives 832 inside-window and 4168 outside-window examples.

TABLE 4.2

RULER `niah_single_1` accuracy (%) for 8K-trained 200M-class checkpoints, split by whether the first token of the needle value lies inside or outside the final 1K tokens of the prompt. The split contains 832 inside-window and 4168 outside-window examples; parentheses give Wilson 95% confidence intervals.

Method	Aggregate	Inside 1K	Outside 1K
FA	97.62%	97.96%	97.55%
	(97.16–98.01)	(96.75–98.72)	(97.04–97.98)
SWA	11.48%	68.99%	0.00%
	(10.63–12.39)	(65.77–72.04)	(0.00–0.09)
IP-TTT	12.16%	65.14%	1.58%
	(11.28–13.10)	(61.84–68.31)	(1.25–2.01)
TTT-E2E	11.56%	69.47%	0.00%
	(10.70–12.48)	(66.26–72.51)	(0.00–0.09)

RESULTS. Table 4.2 shows that FA performs nearly perfectly in both subsets, enabled by its ability to attend directly to the entire context. SWA retrieves 68.99% of the inside-window needles and none of

¹This sweep is not a direct reproduction: we train a separate model from scratch at each context length, whereas Tandon et al. (2025) use context extension, and the experiments also differ substantially in model and data scale. Appendix A presents our attempted reproduction under their context-extension methodology.

the 4168 outside-window needles.² TTT-E2E behaves similarly, reaching 69.47% inside the window and retrieving no outside-window needles. IP-TTT has lower inside-window accuracy, 65.14%, but retrieves 66 of the 4168 outside-window needles, or 1.58%. The aggregate score hides this trade-off: IP-TTT’s limited out-of-window retrieval is partly offset by its lower accuracy when the needle is directly visible.

TAKEAWAY. TTT-E2E shows no evidence of retrieval beyond its direct attention window, agreeing with the retrieval limitation reported by Tandon et al. (2025). IP-TTT, by contrast, retrieves a small but clearly nonzero fraction of out-of-window needles, qualitatively matching the retrieval advantage reported by Feng et al. (2026).³ Its 1.58% accuracy nevertheless remains far below FA and comes with lower inside-window accuracy. Aggregate RULER accuracy therefore obscures a meaningful difference in memory behavior between the methods.

4.3 SYNTHETIC KEY-VALUE RECALL

QUESTION. The pre-training experiments in Section 4.1 test whether TTT can improve language modeling beyond a fixed local attention window. However, the split RULER results in Section 4.2 show only limited out-of-window retrieval for IP-TTT and none for TTT-E2E. Can TTT improve long-range retrieval when retrieval is the only objective optimized during training?

DATASET. We use synthetic associative-recall data to isolate out-of-window retrieval more directly: the model must remember a query–value association after its value has fallen outside the local attention window. This dataset is closely related to Multi-Query Associative Recall (MQAR) (Arora et al., 2023). Each document constructs a fresh table of 4095 disjoint query–value token pairs from an 8192-token vocabulary.⁴ Specifically, a document-specific random number generator permutes all 8190 non-special token IDs; the first 4095 become queries and the remaining 4095 become their values in matching order. This independently randomizes both the query–value mapping and which pairs occur frequently in every document, so token identities have no global meaning across documents and the model must infer the associations from the current document.

Each 8K sequence alternates query and value tokens, $(q_1, v_1, q_2, v_2, \dots)$. Queries are sampled with replacement from a Zipf distribution proportional to $1/\text{rank}$, so common queries repeat often enough to provide many recall targets; the value paired with a query is deterministic within a document. The loss mask scores only value predictions, so at each query position the model must predict the following value token. We exclude first occurrences from Table 4.3. Since every occurrence of a query has the same associated value, the prediction target is unambiguous. For distance classification, we use the nearest preceding occurrence. Thus, a query is classified as outside the window only when no previous occurrence of its value lies within the last 1024 tokens.

BASELINES. Because this controlled task is substantially smaller than language pre-training, the base architecture is a 14M-class decoder-only Transformer with 4 layers, hidden size 512, 4 attention heads, SwiGLU MLP width 1024, tied embeddings, and an 8192-token vocabulary. The data is generated directly

²Zero outside-window retrieval is not a theoretical requirement for SWA: stacking local-attention layers expands the receptive field beyond a single window. However, this access is indirect, requiring information to propagate through intermediate representations. Longformer similarly motivates supplementing local windows with task-specific global attention for long-document downstream tasks (Beltagy, Peters and Cohan, 2020).

³This is not a direct reproduction: their results aggregate multiple RULER tasks on 4B–14B continually pretrained models, whereas ours concern one needle-in-a-haystack task on 200M-class models trained from scratch.

⁴The implementation reserves IDs 0 and 1 as beginning-of-sequence and end-of-sequence identifiers and excludes them from pair generation, leaving exactly 8190 usable IDs for 4095 disjoint pairs. These identifiers are never inserted into the generated sequences.

as token IDs and therefore does not require a tokenizer. The FA baseline can attend directly to the previous occurrence of a query–value pair anywhere in the 8K sequence, making it an oracle-style upper baseline. The SWA baseline has the same model shape, but each token can attend only to the most recent 1024 tokens.

TTT-E2E & IN-PLACE TTT. Both TTT methods use the same 1024-token local-attention window as the SWA baseline. The standard IP-TTT variant updates the MLP down projections in layers 2 and 4, while the all-layer variant updates all four layers. They use inner-loop learning rate $\eta = 3$ and 1024-token update chunks. TTT-E2E adds one prime MLP of width 1024 in the last layer and uses 1024-token chunks. The total trainable parameter counts are 14.69M for FA and SWA, 15.75M for standard and all-layer IP-TTT, and 16.26M for TTT-E2E. The corresponding fast-weight counts are 1.05M for standard IP-TTT, 2.10M for all-layer IP-TTT, and 1.57M for TTT-E2E. These rows are therefore architectural ablations rather than a strictly parameter-matched comparison.

TRAINING. We train every model on the same synthetic key–value distribution of 2,097,152 indexed documents, reserving 5% of document indices for evaluation. Each reported checkpoint has processed 20.97B tokens, corresponding to 2,560,000 sequences of length 8192, so the training-token budget is matched across methods. Training uses 10,000 optimizer steps with a global batch size of 256 sequences. The outer optimizer is AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, peak learning rate 3×10^{-3} , cosine decay to 10^{-5} , 10% linear warmup, weight decay 0.1, and gradient clipping at 1.0. GPU-hour accounting for these five reported runs is summarized in Appendix Table B.2. We evaluate the first 256 documents from the held-out split, containing 136,823 repeated queries whose previous value is outside the window and 598,508 whose previous value is inside it.⁵ For evaluation, we take the argmax vocabulary prediction at each scored query position and compare it with the true following value token. Table 4.3 reports top-1 accuracy over these scored value predictions. Because each document contributes many correlated predictions and each configuration uses one training seed, we do not report confidence intervals or interpret small differences between configurations as reliable.

TABLE 4.3

Top-1 value-prediction accuracy for repeated queries in the synthetic associative-recall task at 8K context. A query is inside 1K if the value following its nearest preceding occurrence lies within the local attention window, and outside 1K otherwise. Estimates are rounded to two decimal places, with additional precision for values below 0.01%.

Model	Outside 1K	Inside 1K
FA	100.00%	100.00%
SWA	0.13%	99.99%
IP-TTT	61.31%	93.40%
IP-TTT, all layers	36.91%	94.02%
TTT-E2E, last layer	< 0.01%	> 99.99%

RESULTS. Table 4.3 shows that FA recalls essentially every repeated association, regardless of its distance, while SWA is nearly perfect only when the previous value remains inside its local window. Once the previous value falls outside that window, SWA accuracy drops to nearly zero. Its 0.13% out-of-window accuracy is above random guessing, but the aggregate result does not identify the mechanism; propagation

⁵Each document is generated independently from its document index and the fixed synthetic-data seed, so this fixed prefix is not selected according to model outcomes.

through successive local-attention layers is one possible explanation. The standard IP-TTT configuration recovers about 61% of the out-of-window associations, showing that this IP-TTT configuration can learn substantial long-range associative recall when trained directly for it. However, its in-window accuracy falls to about 93%, so the strong out-of-window recall in the active IP-TTT configuration coincides with weaker local recall. Applying IP-TTT to all four layers reduces out-of-window recall to about 37%. The IP-TTT parameterization permits each additional layer to learn a zero update, so the substantially lower point estimate with TTT in every layer is surprising. TTT-E2E remains nearly perfect inside the attention window but recovers fewer than 0.01% of out-of-window associations.

TAKEAWAY. Among the configurations tested, IP-TTT learns substantial associative recall beyond its attention window, whereas TTT-E2E gains little. This ability is incomplete and comes at the cost of worse in-window recall. Because we do not evaluate these checkpoints with TTT disabled, this experiment does not by itself isolate how much of the difference is caused by the active update rather than by training with the update dynamics. Updating every layer reduces recall despite the model’s ability to represent zero updates, suggesting that additional TTT layers can complicate optimization.

4.4 RETROFITTING TTT TO A PRETRAINED SWA CHECKPOINT

QUESTION. IP-TTT is designed primarily as a drop-in enhancement for pretrained models, whereas TTT-E2E normally meta-learns its initialization from scratch. Can either method improve an ordinary pretrained SWA model using only a small continuation budget? We test this by attaching each TTT method to the same checkpoint and comparing it with an ordinary continuation of that checkpoint.

STARTING CHECKPOINT AND METHODS. All runs start from the final 8K SWA-1K checkpoint from Section 4.1. For IP-TTT, the MLP down-projections in layers 7 and 13 become fast weights. These projections retain their pretrained values; the newly added target convolution is zero-initialized and its projection is initialized to a near-zero random diagonal. For TTT-E2E, prime MLP sublayers are inserted before the ordinary MLPs in the final three Transformer blocks. Unlike the parameter-matched from-scratch TTT-E2E model, this retrofit retains the SWA checkpoint’s 2816-wide ordinary MLPs and adds prime MLPs of the same width, giving 225.77M parameters in total. The prime sublayers use freshly initialized unit-scale RMSNorms. The gate and up projections of each prime MLP are randomly initialized, while its down projection is initialized to zero. Consequently, each added residual sublayer initially outputs zero, and before the first inner-loop update the augmented model exactly preserves the function of the SWA checkpoint. This initialization does not suppress the inner update: the random gate and up projections produce nonzero hidden activations, so the language-modeling loss generally has a nonzero gradient with respect to the zero down projection. After the first chunk, the updated down projection can therefore alter subsequent predictions.

TRAINING. We continue each model for 200M tokens, equal to 5% of the original 4B-token pre-training budget. The ordinary SWA continuation serves as the control for improvements caused by additional training alone. All three runs use the same FineWeb-Edu data, 8K sequence length, and token batch of 524,288 tokens. The TTT methods retain the 1024-token chunks and inner-loop settings of the corresponding from-scratch runs: IP-TTT uses $\eta = 3$, while TTT-E2E uses learning rate 1.0 and inner-loop gradient clipping at 1.0. We load only the model parameters from the SWA checkpoint and restart the outer optimizer. The continuation lasts 382 optimizer steps and uses AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, weight decay 0.1, and gradient clipping at 1.0. The outer learning rate increases linearly from zero to 8×10^{-5} over the first 38 steps, then follows a cosine decay to 10^{-5} by the end of training. We run one training seed per continuation configuration. We evaluate held-out next-token loss on the same validation documents used in the pre-training sweep.

TABLE 4.4

Held-out next-token loss at 8K context. The from-scratch results are those already presented in Table 4.1 and Figure 4.1. INITIAL is measured before the continual training starts. FINAL is measured after 200M continuation tokens, with the update active for the TTT methods.

Method	From scratch	Continuation from SWA			
	FINAL	INITIAL	INITIAL TTT-OFF	FINAL	FINAL TTT-OFF
SWA	2.011	2.011	–	2.011	–
IP-TTT	1.991	2.011	2.011	2.010	2.011
TTT-E2E	1.994	2.923	2.011	2.145	2.172

TABLE 4.5

Maximum logged pre-clipping outer-gradient norm during training.

Method	From scratch	Continuation from SWA
SWA	4.93	0.087
IP-TTT	6.00	4.66
TTT-E2E	4.05	6.64×10^6

RESULTS. Table 4.4 compares the continuation runs with the corresponding models trained from scratch (the from-scratch performance was already seen in Figure 4.1). For TTT-E2E, this is not an architecture-matched comparison: the retrofit retains the 2816-wide SWA backbone and adds capacity, whereas the from-scratch model reallocates capacity to remain parameter-matched. The SWA control finishes at the same loss as its starting checkpoint, so the additional 200M tokens alone do not improve held-out loss at the reported precision. IP-TTT finishes at 2.010, only 0.001 below both its initial loss and the final SWA loss. This gain is marginal, and IP-TTT remains substantially worse than its 1.991 from-scratch counterpart. TTT-E2E recovers strongly from its poor initial loss of 2.923 but finishes at 2.145, still 0.134 worse than the shared SWA checkpoint and 0.151 worse than TTT-E2E trained from scratch.

CONTRIBUTION OF THE ACTIVE UPDATE. Before continuation, enabling TTT-E2E’s untrained inner update raises held-out loss from 2.011 with TTT-OFF to 2.923. By the end of continuation, this ordering reverses. For the final checkpoint, enabling the update lowers loss by 0.027, from 2.172 to 2.145. However, even with the update enabled, the retrofitted model remains worse than the original checkpoint. For the final IP-TTT checkpoint, enabling the update lowers loss by only 0.001, from 2.011 to 2.010.

OPTIMIZATION STABILITY. TTT-E2E’s continuation is accompanied by a severe transient instability. As shown in Table 4.5, the maximum logged pre-clipping outer-gradient norm for TTT-E2E is 6.64×10^6 , six orders of magnitude larger than for either IP-TTT or SWA. The run nevertheless recovers and completes training, likely thanks to the applied gradient clipping.

TAKEAWAY. Within this 200M-token continuation budget, neither retrofit reproduces its from-scratch performance or meaningfully improves on the pretrained SWA checkpoint. IP-TTT provides only a marginal 0.001 gain, while TTT-E2E learns to turn an initially harmful update into a beneficial one but still finishes substantially worse than the checkpoint and experiences extreme transient gradient growth. The experiment therefore provides no evidence that either method is an effective retrofit under this training setup, although TTT-E2E’s recovery shows that its newly added update can be learned during continuation.

The comparison is limited to one 200M-class SWA model and a continuation budget much smaller than those used in the primary pretrained-model experiments of Feng et al. (2026).

CHAPTER 5

CONCLUSION

RESEARCH QUESTIONS AND MAIN ANSWERS. This thesis asked which TTT implementation performs better under a matched from-scratch long-context training protocol, whether any advantage comes from the static substrate or the test-time update, to what extent each method supports explicit retrieval beyond its attention window, and whether either TTT method can improve a pretrained SWA checkpoint under a small continuation budget. At 200M-class scale, the matched sweep does not support a reliable ranking between IP-TTT and TTT-E2E. Both methods outperform SWA at longer contexts and recover part of its loss gap to FA; at 8K, IP-TTT has a 0.003 lower point-estimate loss than TTT-E2E, but the single training seed and the methods’ different architectures prevent interpreting this gap as a general ranking. For both trained checkpoints, the active update is necessary to outperform SWA at longer contexts: disabling it makes each method worse than SWA. Aggregate RULER scores are similar after language pre-training, but splitting by needle position reveals that IP-TTT retrieves 1.58% of out-of-window needles, while SWA and TTT-E2E retrieve no out-of-window needles among the 4168 evaluated examples. In the single-seed synthetic experiment, whose method configurations are not strictly matched in architecture or parameter count, IP-TTT recovers 61.3% of out-of-window associations, whereas the tested TTT-E2E configuration recovers less than 0.01%. Finally, the continuation experiment provides no evidence that either method is an effective retrofit within a 200M-token budget: IP-TTT improves only marginally, while TTT-E2E learns a beneficial update but remains substantially worse than the pretrained SWA checkpoint and experiences extreme transient gradients, indicating stability issues.

INTERPRETATION. Both methods can be understood as adding a compressed state alongside local attention. Their active fast-weight updates improve later-position language-modeling loss, consistent with carrying useful predictive information after tokens leave the SWA window; unlike FA, however, they cannot retain direct access to every preceding representation. This distinction is consistent with the main pre-training result: the updates recover part of the loss caused by restricting attention, but neither method closes the gap to FA. The degradation of both active TTT methods with token position further suggests that their adaptive state does not preserve all information that remains available through global attention.

The comparison also indicates that fast-weight count alone does not determine performance. TTT-E2E exposes approximately 21.1M fast weights across its three prime MLPs, compared with 5.77M in the two IP-TTT down-projections, or about 3.66 times as many, yet the two methods achieve similar language-modeling loss. One possible explanation is that IP-TTT’s outer-product update explicitly writes associations between MLP activations and token-derived targets, making its parameterization well suited to key–value recall. TTT-E2E instead updates its fast weights using the final next-token loss, which may favor predictive information that reduces average language-modeling loss without supporting exact retrieval. The synthetic ablation likewise shows that updating more IP-TTT layers reduces recall despite

the model’s ability to represent zero updates, suggesting that additional TTT layers can complicate optimization. These experiments do not isolate which of these factors causes the retrieval difference, so this explanation remains a hypothesis.

Finally, the methods present different practical trade-offs. TTT-E2E directly aligns its update with the language-modeling objective, but requires second-order meta-training and a jointly learned initialization; attaching untrained prime MLPs to an existing checkpoint therefore creates a difficult and, in our experiment, unstable adaptation problem. IP-TTT uses a closed-form local update, retains compatibility with FlashAttention kernels, and was designed for retrofitting, but its marginal continuation gain shows that these implementation properties do not guarantee effective adaptation under a small token budget. Because we do not report end-to-end throughput, latency, or memory use, they also do not by themselves establish a systems-level advantage. The choice between the methods therefore depends not only on final loss, but also on the desired memory behavior, available training regime, and systems constraints.

LIMITATIONS. Several limitations constrain the scope of these conclusions. The language-modeling results cover only 200M parameter models trained for 4B tokens on a subset of FineWeb-Edu, with context lengths up to 8K. They therefore do not establish whether the relative performance of the methods changes with model size, training duration, data domain, or longer contexts. Moreover, the main sweep contains one completed run per method and configuration. Although evaluation uncertainty is measured where possible, variation across training seeds is not. Small differences in loss, in particular the difference between IP-TTT and TTT-E2E at 8K, should consequently not be interpreted as a reliable ranking.¹

The comparison also fixes particular implementations rather than exhaustively optimizing each method. Both methods use a 1K attention window, and their update placement and other main design choices largely follow the respective papers instead of being selected through a shared hyperparameter search. The parameter counts are not perfectly matched: IP-TTT has 6.9% more parameters than FA and SWA, whereas TTT-E2E remains parameter-matched by reallocating capacity from its ordinary MLPs to its prime MLPs. In addition, the main experiment places IP-TTT in its secondary SWA/from-scratch regime rather than the larger pretrained-model setting that motivates the method. The controlled comparison is therefore informative about the closest shared regime in which both methods can be instantiated, but not about every regime in which either method may be used.

The retrieval evaluation is similarly narrow. The natural-language diagnostic uses only the easiest single-needle RULER task at 8K and does not test harder retrieval, aggregation, reasoning, or extrapolation beyond the training length. Because RULER was not evaluated with the TTT updates disabled, the observed IP-TTT advantage outside the window cannot be attributed specifically to the active update. The synthetic experiment isolates associative recall more directly, but uses smaller models, and its architectural ablations are not strictly matched in parameter count. It also uses one training seed per configuration and does not isolate the active update with a TTT-off evaluation.

Finally, the retrofit experiment starts from a single SWA checkpoint and uses only 200M continuation tokens, far less than the budgets used in the primary pretrained-model experiments of Feng et al. (2026). Its negative result therefore applies only to this small-budget setting. The thesis also focuses on predictive quality and retrieval capability rather than providing a controlled end-to-end comparison of training throughput, inference throughput, and memory use. These systems properties are important to the practical value of TTT, especially because the implementations differ in their support for FlashAttention kernels.

¹Repeating the full sweep across seeds is expensive; Appendix Table B.2 reports the GPU time required for the minimal set of runs underlying the main results.

FUTURE WORK. First, future experiments should identify why our TTT-E2E results are more modest than the long-context scaling reported by Tandon et al. (2025). The two settings differ simultaneously in model size, training-token budget, attention-window size, data, and whether context length is varied during pre-training or through context extension. Controlled sweeps over these factors are needed to determine whether TTT-E2E’s scaling advantage emerges only beyond a particular model or data scale, depends on the ratio between the attention window and sequence length, or is specific to the context-extension protocol.

Second, the mechanism behind the different memory behavior of the two methods remains unresolved. TTT-E2E improves language-modeling loss without retrieving any out-of-window RULER needles and learns almost no out-of-window associative recall. IP-TTT retrieves a small fraction of out-of-window RULER needles after language pre-training and learns substantial recall when trained directly for it. Probing and intervening on the fast-weight state could reveal what information each update preserves, whether TTT-E2E stores a compressed predictive representation rather than explicit key–value associations, and why applying IP-TTT to more layers reduces recall despite permitting zero updates.

Third, making TTT-E2E an effective retrofit remains an open problem. Our zero-initialized prime branch preserves the pretrained model before the first inner-loop update, but it does not prevent that update from being immediately harmful, and continuation training exhibits extreme transient gradient growth. Promising directions include initially gating off the prime branch or inner update and enabling it gradually, training the new branch before unfreezing the backbone, and testing longer continuation budgets. These approaches could enable adding TTT-E2E to larger pretrained local- or hybrid-attention models.

Finally, predictive quality should be evaluated together with systems cost. A controlled comparison of training throughput, prefill and decoding latency, memory use, and achievable context length would quantify the practical consequences of TTT-E2E’s second-order training and IP-TTT’s compatibility with fused attention kernels such as FlashAttention.

CLOSING PERSPECTIVE. Neither tested TTT implementation is a universal winner or a complete substitute for full attention. Both recover some long-context language-modeling quality lost to local attention, while the tested IP-TTT configuration also learns substantial explicit associative recall that the tested TTT-E2E configuration does not. More broadly, average language-modeling loss alone does not reveal what an adaptive state remembers or whether it supports explicit retrieval. Future TTT methods should therefore be judged by predictive quality, memory behavior, learning stability, and measured deployment efficiency in their intended training regime.

BIBLIOGRAPHY

- Vaswani, Ashish et al. (Dec. 2017). *Attention Is All You Need*. DOI: 10.48550/arXiv.1706.03762. arXiv: 1706.03762 [cs]. (Visited on 29th Apr. 2026).
- Dao, Tri and Albert Gu (2024). *Transformers Are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality*. arXiv: 2405.21060 [cs.LG]. URL: <https://arxiv.org/abs/2405.21060>.
- Yang, Songlin, Jan Kautz and Ali Hatamizadeh (Mar. 2025). *Gated Delta Networks: Improving Mamba2 with Delta Rule*. DOI: 10.48550/arXiv.2412.06464. arXiv: 2412.06464 [cs]. (Visited on 29th Apr. 2026).
- Sun, Yu et al. (Aug. 2025). *Learning to (Learn at Test Time): RNNs with Expressive Hidden States*. DOI: 10.48550/arXiv.2407.04620. arXiv: 2407.04620 [cs]. (Visited on 29th Apr. 2026).
- Tandon, Arnub et al. (Dec. 2025). *End-to-End Test-Time Training for Long Context*. DOI: 10.48550/arXiv.2512.23675. arXiv: 2512.23675 [cs]. (Visited on 25th Feb. 2026).
- Feng, Guhao et al. (Apr. 2026). *In-Place Test-Time Training*. DOI: 10.48550/arXiv.2604.06169. arXiv: 2604.06169 [cs]. (Visited on 10th Apr. 2026).
- Penedo, Guilherme et al. (2024). *The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale*. arXiv: 2406.17557 [cs.CL]. URL: <https://arxiv.org/abs/2406.17557>.
- Beltagy, Iz, Matthew E. Peters and Arman Cohan (2020). *Longformer: The Long-Document Transformer*. arXiv: 2004.05150 [cs.CL]. URL: <https://arxiv.org/abs/2004.05150>.
- Team, Gemma, Morgane Riviere et al. (2024). *Gemma 2: Improving Open Language Models at a Practical Size*. arXiv: 2408.00118 [cs.CL]. URL: <https://arxiv.org/abs/2408.00118>.
- Team, Gemma, Aishwarya Kamath et al. (2025). *Gemma 3 Technical Report*. arXiv: 2503.19786 [cs.CL]. URL: <https://arxiv.org/abs/2503.19786>.
- Team, Kimi et al. (2025). *Kimi Linear: An Expressive, Efficient Attention Architecture*. arXiv: 2510.26692 [cs.CL]. URL: <https://arxiv.org/abs/2510.26692>.
- Kimi K3 Tech Blog: Open Frontier Intelligence* (2026). URL: <https://www.kimi.com/blog/kimi-k3> (visited on 21st July 2026).
- Zhang, Tianyuan et al. (May 2025). *Test-Time Training Done Right*. DOI: 10.48550/arXiv.2505.23884. arXiv: 2505.23884 [cs]. (Visited on 29th Apr. 2026).
- Shah, Jay et al. (2024). *FlashAttention-3: Fast and Accurate Attention with Asynchrony and Low-precision*. arXiv: 2407.08608 [cs.LG]. URL: <https://arxiv.org/abs/2407.08608>.
- Hsieh, Cheng-Ping et al. (Aug. 2024). *RULER: What's the Real Context Size of Your Long-Context Language Models?* DOI: 10.48550/arXiv.2404.06654. arXiv: 2404.06654 [cs]. (Visited on 5th Mar. 2026).
- Arora, Simran et al. (2023). *Zoology: Measuring and Improving Recall in Efficient Language Models*. arXiv: 2312.04927 [cs.CL]. URL: <https://arxiv.org/abs/2312.04927>.
- Shazeer, Noam (2020). *GLU Variants Improve Transformer*. arXiv: 2002.05202 [cs.LG]. URL: <https://arxiv.org/abs/2002.05202>.

- Zhang, Biao and Rico Sennrich (2019). *Root Mean Square Layer Normalization*. arXiv: 1910.07467 [cs.LG]. URL: <https://arxiv.org/abs/1910.07467>.
- Grattafiori, Aaron et al. (2024). *The Llama 3 Herd of Models*. arXiv: 2407.21783 [cs.AI]. URL: <https://arxiv.org/abs/2407.21783>.
- Henry, Alex et al. (2020). *Query-Key Normalization for Transformers*. arXiv: 2010.04245 [cs.CL]. URL: <https://arxiv.org/abs/2010.04245>.
- Hoffmann, Jordan et al. (2022). *Training Compute-Optimal Large Language Models*. arXiv: 2203.15556 [cs.CL]. URL: <https://arxiv.org/abs/2203.15556>.
- Loshchilov, Ilya and Frank Hutter (2019). *Decoupled Weight Decay Regularization*. arXiv: 1711.05101 [cs.LG]. URL: <https://arxiv.org/abs/1711.05101>.
- Su, Jianlin et al. (2023). *RoFormer: Enhanced Transformer with Rotary Position Embedding*. arXiv: 2104.09864 [cs.CL]. URL: <https://arxiv.org/abs/2104.09864>.
- Li, Jeffrey et al. (2025). *DataComp-LM: In search of the next generation of training sets for language models*. arXiv: 2406.11794 [cs.LG]. URL: <https://arxiv.org/abs/2406.11794>.
- Rae, Jack W. et al. (2019). *Compressive Transformers for Long-Range Sequence Modelling*. arXiv: 1911.05507 [cs.LG]. URL: <https://arxiv.org/abs/1911.05507>.

APPENDIX A

ADDITIONAL EXPERIMENTS

A.1 ATTEMPTED REPRODUCTION OF TTT-E2E CONTEXT-EXTENSION SCALING

PURPOSE. Before designing the pre-training length sweep in Section 4.1, we reproduced the methodology behind the headline result of Tandon et al. (2025, Fig. 1). That experiment asks the same question as our headline experiment in Section 4.1, but uses a different methodology.¹ The characteristic result is a separation in scaling geometry: TTT-E2E maintains an approximately constant advantage over FA as context length grows, whereas SWA becomes progressively worse than FA. Our goal was to test whether the same geometry appears in our setting, while adding IP-TTT as an additional comparison.

MODEL AND TRAINING RECIPE. Our models have 12 Transformer blocks, hidden size 768, 12 attention heads, gated-MLP width 2048, tied embeddings, RMSNorm, RoPE (Su et al., 2023), and the Llama-3 vocabulary, giving a total size of approximately 183M parameters. TTT-E2E reduces the ordinary and prime MLP widths to 1664 so that adding prime MLPs in the final three blocks does not increase the total parameter count. IP-TTT retains the ordinary MLP shape, constructs its update target with a depthwise 1D convolution and learned projection, and updates the MLP down-projections in zero-indexed blocks {5, 11} (layers 6 and 12) after each 1024-token chunk.

We pre-train for 2.5B tokens at 8K context. DataComp for Language Models Baseline (Li et al., 2025) is filtered to documents containing at least 8K tokens. The optimizer is AdamW with peak learning rate 3×10^{-3} , cosine decay to 10^{-5} , 10% warmup, weight decay 0.1, and gradient clipping at 1.0. Because an 8K SWA window covers the complete pre-training sequence, FA and SWA are identical during this stage and share the same pre-training checkpoint.

For each extension length, we fine-tune for 125M PG-19 (Rae et al., 2019) tokens, equal to 5% of the pre-training budget, using peak learning rate 4×10^{-4} and a restarted cosine schedule. Only FA increases its RoPE base and attention span with T . The local-attention methods retain the pre-training RoPE base and an 8K window because their attention layers never encounter larger relative offsets.

EVALUATION. We report bits per byte (BPB) on a held-out PG-19 partition. Each checkpoint is evaluated at the context length at which it was fine-tuned, so this is a training-length scaling experiment rather than an extrapolation test. For both TTT methods, we also evaluate the same checkpoint with its

¹Instead of pre-training a model from scratch for each context length, it uses one checkpoint pre-trained on 8K-token sequences and extends it by fine-tuning on sequences of 16K to 128K tokens.

inner-loop update disabled. This TTT-OFF diagnostic separates the quality of the learned static substrate from the gain produced by the active test-time update.

DIFFERENCES FROM THE ORIGINAL EXPERIMENT. The structure of the experiment mirrors Tandon et al. (2025), but its scale does not. Their headline result uses a 3B-parameter model trained with 164B tokens; ours uses a 183M-class model pre-trained for 2.5B tokens and extension-fine-tuned for 125M tokens per length. We also add IP-TTT, which was not part of the original TTT-E2E comparison and is evaluated here in its secondary from-scratch SWA regime rather than its primary drop-in/full-attention regime.

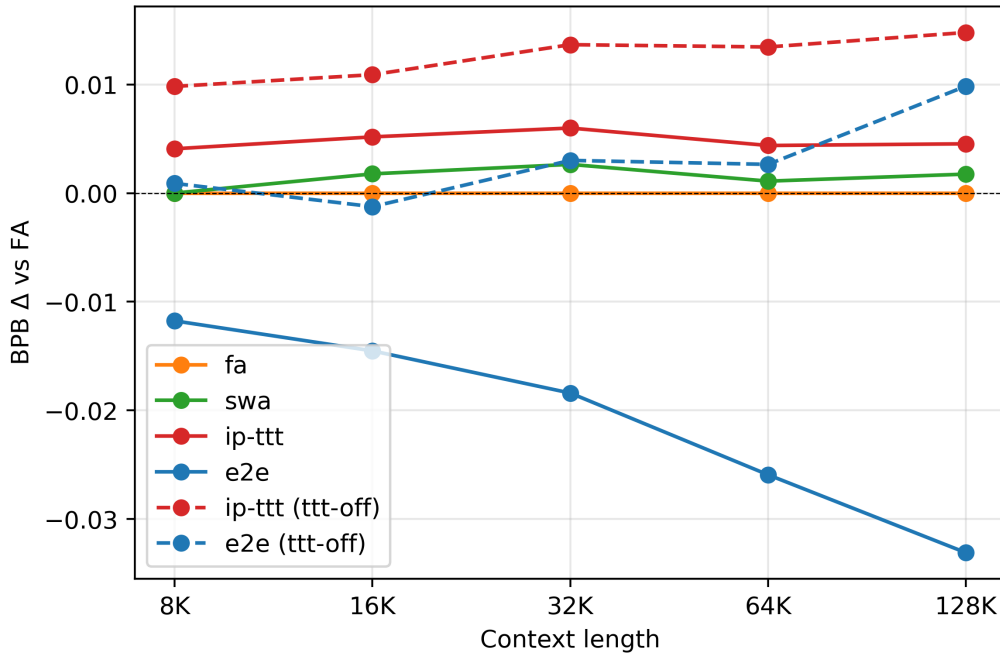


FIGURE A.1

Held-out PG-19 BPB Δ relative to FA for the context-extension experiment. The 8K markers are pre-training checkpoints; the longer-context markers are separately fine-tuned models evaluated at the T they were extended to. The TTT-OFF curves evaluate the same TTT-method checkpoint with the inner-loop update disabled at test time, exposing the static substrate of each TTT method.

A.1.1 RESULTS AND INTERPRETATION

THE REQUIRED BASELINE SEPARATION IS ABSENT. Figure A.1 does not reproduce the scaling geometry of the original headline result. FA and SWA remain within 0.003 BPB of each other from 8K through 128K. In the original experiment, SWA becomes progressively worse than FA as T grows beyond its fixed window; that separation demonstrates that the task rewards access to context outside the window and creates the deficit that TTT-E2E is intended to recover. Here, allowing FA to attend beyond 8K provides almost no measurable BPB benefit over SWA. The experiment therefore lacks its essential control: held-out PG-19 loss at this model scale and training budget does not reveal a long-context limitation of the 8K SWA baseline.

THE TTT-E2E CURVE IS CONSEQUENTLY AMBIGUOUS. TTT-E2E improves from 0.012 BPB below FA at 8K to 0.033 below FA at 128K. This is numerically favorable, but it is not the result established by the original experiment. Rather than maintaining an approximately constant gap to FA while SWA

deteriorates, TTT-E2E pulls farther below two non-TTT baselines that remain nearly coincident. Because FA itself obtains no advantage from the additional context, the growing TTT-E2E gain cannot be attributed specifically to recovering information that lies outside the SWA window.

WHY THIS EXPERIMENT IS NOT USED AS THE MAIN RESULT. The immediate failure is methodological rather than merely an unfavorable method ranking: with an 8K window, the FA–SWA comparison does not show that this evaluation benefits from longer accessible context. The large gap from the original scale — 183M versus 3B parameters and billions rather than 164B training tokens — is a plausible explanation, but this experiment does not isolate model scale from data volume or optimization. Section A.2 tests one part of this explanation by increasing the extension budget, but does not recover a qualitatively different ordering.

The main experiment in Section 4.1 was designed in response to this failure. It trains a separate model from scratch at context lengths 1K to 8K on FineWeb-Edu and fixes the SWA window at 1K. That narrower window produces a clear FA–SWA separation as training length increases, establishing that the model can benefit from information outside local attention. The two experiments therefore answer different questions: this appendix documents an attempt to reproduce the original context-extension methodology, while the main experiment deliberately changes the methodology to obtain an informative controlled test of long-context language modeling.

A.2 EXTENSION-TOKEN BUDGET ABLATION

PURPOSE. The reproduction attempt in Section A.1 uses 125M PG-19 extension tokens per context length. That is intentionally small compared with the continuation budgets used in the IP-TTT paper, so a natural concern is that the IP-TTT deficit is a low-budget artifact. We test this by increasing only the amount of PG-19 extension data while holding the model family, checkpoints, context lengths, attention windows, and evaluation protocol fixed.

PROTOCOL. We run the ablation at the two lengths, 32K and 128K. For each method, the 125M point is the canonical run from the reproduction attempt. We then continue the same extension recipe to cumulative budgets of 250M, 500M, and 1B PG-19 tokens. The optimizer schedule is restarted for the longer continuation runs in the same style as the canonical extension stage, and BPB is always measured on the held-out PG-19 partition at the same context length as training. The primary experimental variable is therefore the volume of data used for context-extension fine-tuning.

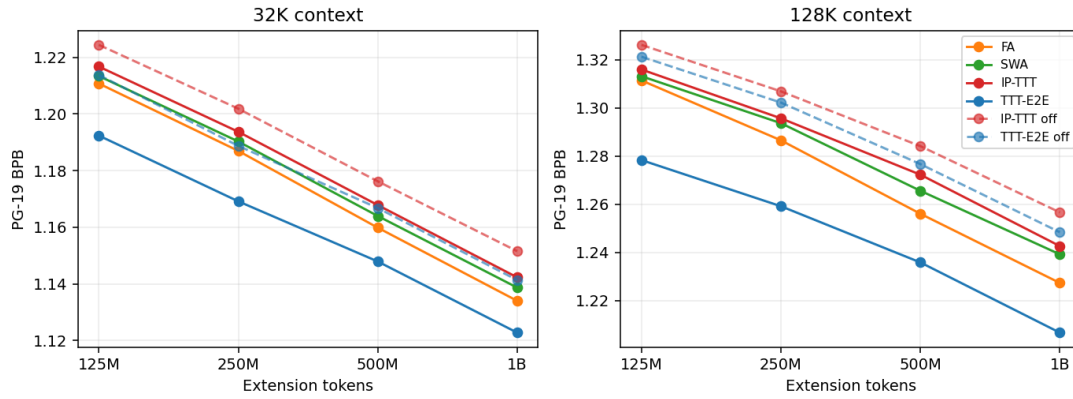


FIGURE A.2

Held-out PG-19 BPB versus cumulative extension tokens at 32K and 128K context. All methods’ point estimates improve with more PG-19 extension data. TTT-E2E stays lowest, while the small differences between IP-TTT and SWA do not support a confident ranking without a paired uncertainty analysis.

Figure A.2 shows that additional extension data improves all methods without changing the point-estimate ordering. At both context lengths, the TTT-E2E point estimate remains below FA, SWA, and IP-TTT throughout the sweep. IP-TTT and SWA remain numerically close, so these evaluations do not establish an ordering between them. Within this 183M-model regime, there is therefore no evidence that a larger extension budget produces a clear IP-TTT advantage; such an advantage may require larger models, much larger continuation budgets, or the paper’s primary drop-in/full-attention setting.

APPENDIX B

IMPLEMENTATION

The FA, SWA, and TTT-E2E runs use the authors’ official JAX/Flax implementation of TTT-E2E (Tandon et al., 2025), available at <https://github.com/test-time-training/e2e>, as their starting point. We ported the authors’ official PyTorch implementation of In-Place TTT (Feng et al., 2026), available at <https://github.com/ByteDance-Seed/In-Place-TTT>, into the same JAX/Flax codebase to permit a controlled comparison.

The main changes to the TTT-E2E codebase were:

- adding a Modal-based training pipeline;
- replacing the original pre-tokenized Zarr/GCS input pipeline with configurable Hugging Face dataset loading, tokenization, filtering, and train-validation splitting;
- integrating the JAX port of IP-TTT, including its target construction and closed-form fast-weight update, so that all four methods could share the same training and evaluation stack; and
- extending evaluation and checkpoint handling with periodic evaluation, bits-per-byte and confidence-interval reporting, TTT-on/off comparisons, RULER evaluation, and warm-start experiments from existing checkpoints.

Training and evaluation ran on Modal using A100, H100, H200, and B200 GPUs. The total cost of the cloud-based compute was \$9,306, as summarized in Table B.1. This total includes exploratory, failed, and superseded runs; most of the compute did not contribute directly to the final experiments reported in this thesis.

TABLE B.1

Modal compute used by thesis-related applications. GPU-hours are approximated based on the hourly rates listed on the Modal website.

Resource	Approx. GPU-hours	Usage cost (USD)
H200	1,283	\$5,826
B200	476	\$2,974
H100	46	\$181
A100 80GB	3	\$8
CPU and memory	—	\$317
Total	1,808	\$9,306

To separate the cost of reproducing the reported results from the total development cost, Table B.2 counts the full checkpoint lineage of each training run that contributes to the main pre-training sweep or synthetic key-value recall table. It excludes exploratory runs, unsuccessful independent attempts, the experiments in the appendix, and evaluation-only checkpoint reloads, but includes interrupted segments whose checkpoints were resumed to produce a reported result. The table is GPU-use accounting rather than a controlled speed benchmark: the measurements include run-level overhead and compare implementations with different attention kernels and training dynamics.

TABLE B.2

GPU-hour accounting from measured Weights & Biases runtimes for the minimal set of training runs used in the main pre-training sweep and synthetic key-value recall experiment. The per-run column gives the observed range within each row.

Experiment	Method	Runs	Hours per run	Total GPU-hours
200M pre-training sweep	FA	4	6.4–9.2	31.3
	SWA	4	15.2–20.1	75.1
	IP-TTT	4	10.5–49.6	108.4
	TTT-E2E	4	11.1–13.7	51.3
	Subtotal	16	—	266.1
Synthetic key-value recall	FA	1	4.6	4.6
	SWA	1	15.1	15.1
	IP-TTT variants	2	11.3–12.7	24.0
	TTT-E2E	1	10.1	10.1
	Subtotal	5	—	53.8
Combined		21	—	319.9

APPENDIX C

USE OF AI TOOLS

AI tools were used as writing and coding assistance during this work, in particular to help improve the style and structure of the text. The paper selection, technical ideas, experimental design, interpretation of results, and final responsibility for the code, results, and text remain the author's.