ADAWRITE: ADAPTIVE WRITE DEPTH ROUTING FOR TEST-TIME TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Test-time training (TTT) models retain information from long contexts in fast weights that are updated during inference. Common chunkwise implementations process each chunk at full depth, even though the benefit of deeper processing can vary. We introduce AdaWrite, a parameter-efficient method that reduces inference computation during context processing, or prefill. AdaWrite learns where each updating chunk stops in a frozen TTT backbone. An early exit skips deeper computation and memory updates while leaving skipped layers' fast states unchanged. We first train intermediate readouts, then learn the routing policy with a detached full-depth teacher as a reference and a compute penalty. Our main experiments on LaCT cover retrieval, document understanding, and language modeling. On S-NIAH-1, AdaWrite substantially reduces analytical prefill FLOPs and improves retrieval beyond the training context length. The router concentrates deeper updates near the needle without needle supervision. On other retrieval tasks and natural text, the same policy uses more layers and achieves prediction quality close to the full-depth baseline. An additional study on TTT-E2E shows modest computational savings with a different memory update rule.

1 INTRODUCTION

Long sequences provide context for later predictions, but useful information may be interspersed with repeated or redundant text (Ainslie et al., 2023). Test-time training (TTT) carries this context in fixed-size fast weights (Sun et al., 2025). Recent work introduces more expressive update rules (Behrouz et al., 2025b; 2026), scalable chunkwise computation in LaCT (Zhang et al., 2026), and end-to-end long-context training in TTT-E2E (Tandon et al., 2025). In LaCT and TTT-E2E, every updating chunk executes all stateful layers. Each chunk therefore receives full write depth, even though the benefit of deeper processing may vary.

We introduce AdaWrite to reduce prefill computation by learning where each updating chunk stops in a frozen TTT backbone. Early exits skip deeper computation and preserve skipped layers' incoming fast states. We train the policy with language-model supervision and a compute penalty to balance prediction quality and computation.

Adaptive depth allocates computation for prediction (Graves, 2017; Elbayad et al., 2020; Schuster et al., 2022). In TTT, a stopping decision also controls the persistent memory available to later chunks (Figure 1). Our main configurations add at most 0.2% parameters; forcing CONTINUE everywhere recovers the full-depth path exactly.

On RULER's single-needle retrieval task S-NIAH-1 (Hsieh et al., 2024), LaCT routing improves retrieval beyond the 64K pretraining length with up to 2.21× lower analytical prefill FLOPs (§4). Without needle supervision, deep routes concentrate on chunks containing the needle (§4.4). LaCT's fixed-depth and random-update controls give lower average S-NIAH-1 accuracy. On other inputs, the same policy uses more layers with quality close to its frozen baseline. An additional evaluation on TTT-E2E shows modest savings in its six routable layers.

Our contributions are: (i) AdaWrite, a stopping policy that jointly skips deeper execution and native fast-state updates to reduce prefill computation in frozen TTT models; (ii) a training procedure for frozen backbones that aligns intermediate readouts before learning hard routes with a gate-teacher counterfactual (GTCF) and a survival-weighted compute loss; and (iii) a LaCT study showing re-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

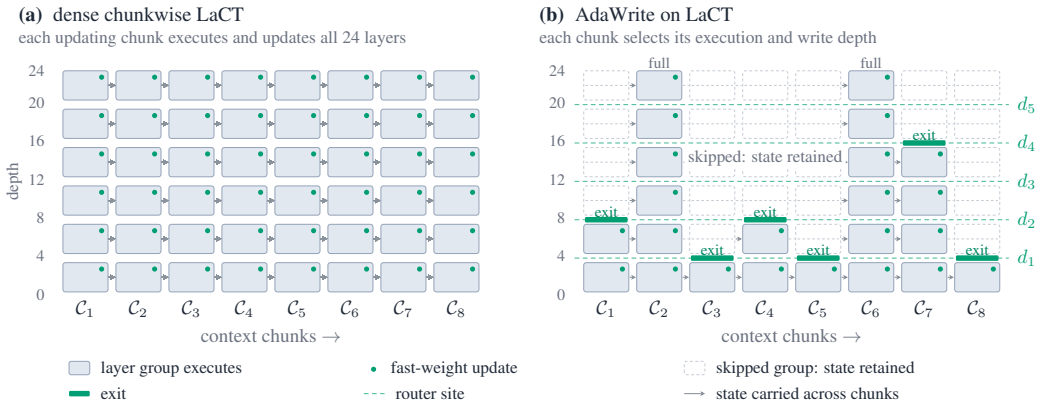


Figure 1: **Adaptive write depth routing with AdaWrite.** (a) Dense chunkwise LaCT: every updating chunk executes all 24 layers and commits their native fast-weight updates (arrows: state flow across chunks). (b) The routed pass (illustrative trace): at each boundary d_j a chunk either CONTINUEs or EXITs; an exit predicts from that depth's readout and preserves every skipped layer's incoming state exactly ($\mathcal{S}_{i,\ell} = \mathcal{S}_{i-1,\ell}$). A single decision sets both the current computation and the deeper memory that later chunks can use.

duced analytical prefill FLOPs and improved S-NIAH-1 retrieval extrapolation, supported by routing analysis and component ablations.

2 Related Work

Test-time training and fast-weight memory. Test-time training began as adaptation to distribution shift (Sun et al., 2020; Wang et al., 2021). EATA selects informative test samples to reduce adaptation cost (Niu et al., 2022). In sequence modeling, TTT instead uses fast weights as memory within a context (Sun et al., 2025), building on fast weight programmers and linear attention (Ba et al., 2016; Schlag et al., 2021). Recent methods improve the update rule, memory architecture, or training objective (Behrouz et al., 2025b;a; Zhang et al., 2026; Tandon et al., 2025; Feng et al., 2026). TNT changes memory organization and chunk size for efficient training (Li et al., 2026). Gdwm allocates gradient steps across context chunks using contextual-utility estimates (Mei et al., 2026). We keep the backbone's chunk boundaries and learn where each chunk stops in the TTT stack. Context distillation (Wang et al., 2026) uses a teacher to shape what fast weights store; our teacher provides a full-depth reference for training the exit gates (§3.4).

Selective and bounded memory. Capacity limits in linear fast-weight memories motivate selective retention (Schlag et al., 2021). For spatial reconstruction, Elastic TTT regularizes updates toward an anchor state to limit forgetting (Ma et al., 2026). Gated linear memories regulate retention and writing within executed layers (Yang et al., 2024; 2025; Hatamizadeh et al., 2026). Other approaches compress or retrieve stored activations (Rae et al., 2019; Wu et al., 2022); Expire-Span learns when to discard them (Sukhbaatar et al., 2021). Our policy skips complete fast-weight transitions, preserving incoming states and saving computation.

Early exit and adaptive depth. Early-exit systems adapt prediction depth in encoders and autoregressive Transformers (Xin et al., 2020; Elbayad et al., 2020; Schuster et al., 2022), with intermediate readouts trained at scale in EE-LLM and LayerSkip (Chen et al., 2024; Elhoushi et al., 2024). Pretrained Transformers also support learned routing and layer skipping (yikun jiang et al., 2024; He et al., 2025; Laitenberger et al., 2026). FlexiDepth adds routers and adapters to frozen LLMs to replace selected layer computations (Luo et al., 2025); PuDDing selects omitted blocks from the prompt (Wee et al., 2025). These methods do not gate native TTT updates. Stateful antecedents include Skip RNN, which learns when to preserve a recurrent state (Campos et al., 2018), and HM-RNN, which learns boundaries for updating, copying, or resetting states across recurrent layers (Chung et al., 2017). Our policy learns a stopping depth for each native TTT chunk.

Conditional computation and learned routing. Sparse mixtures of experts route tokens across width (Shazeer et al., 2017); mixture of depths, recursive Transformers, and CoLT5 allocate computation across depth or branches (Raposo et al., 2024; Bae et al., 2025; Ainslie et al., 2023); and block sparse attention selects context at chunk granularity (Lu et al., 2025; Yuan et al., 2025). We use straight-through gradients for discrete routing (Bengio et al., 2013) and a cost penalty, but align each decision with a complete native TTT transition. This couples the allocation of computation to the memory available to later chunks.

3 Approach

Figure 1 compares full-depth LaCT (a) with AdaWrite (b). Each stopping decision controls both computation and persistent memory: executed layers commit updates, while skipped layers retain their incoming states. After readout alignment, we train routing with a full-depth teacher reference and a compute penalty.

3.1 Preliminaries: Chunkwise Test-Time Training

Let $\mathbf{x} = (x_1, \ldots, x_T)$ be split into ordered chunks $\mathcal{C}_1, \ldots, \mathcal{C}_M$, chunk i holding n_i tokens. A TTT model separates slow parameters θ, fixed during inference, from fast states at a set of stateful blocks $\mathcal{I}_{\text{state}} \subseteq \{0, \ldots, L-1\}$: $\mathcal{S}_i = \{\mathcal{S}_{i,\ell}\}_{\ell \in \mathcal{I}_{\text{state}}}$. Each state contains fast weights and, where present, optimizer state; attention K/V caches are handled separately. A native chunk transition maps $(\mathcal{C}_i, \mathcal{S}_{i-1})$ to representations $\mathbf{H}_i$ and a new state $\mathcal{S}_i$; the current chunk is evaluated with the incoming state before its update becomes visible. We distinguish *execution depth* D_i, the number of backbone blocks run on the routed path, from *write depth* W_i, the number of stateful blocks whose native transition is committed.

The two backbones. We study LaCT and TTT-E2E, two chunkwise TTT backbones with public implementations and different update mechanisms. LaCT (Zhang et al., 2026) aggregates local key–value gradients with per-token learning rates and a stateful optimizer. TTT-E2E (Tandon et al., 2025) applies a chunk-averaged next-token cross-entropy gradient step to fast MLPs in its deep suffix. We preserve their update granularity, inner-loss form, optimizer, and read-before-write schedule (Appendix A); TTT-E2E evaluates its inner gradients through the routed predictor.

3.2 Chunk-Adaptive Write Depth Routing

Sequential stopping policy. Following Figure 1(b), we place routers at selected layer boundaries, called routing sites. Index blocks from zero and let $0 < d_1 < \cdots < d_K < L$, with $d_{K+1}{=}L$. Site j follows the first d_j blocks and controls the next group $\mathcal{G}_j = [d_j, d_{j+1})$; $[0, d_1)$ is an unconditional stem. On an eligible chunk (nonterminal, with a native update), Exit predicts at the boundary and stops its stateful path, whereas Continue executes $\mathcal{G}_j$ and advances. This is a sequential stopping rule, not independent exit classification.

Why route at native update granularity. A native update aggregates gradients over a whole chunk, so all its tokens share one route. Each continued group executes a complete native transition (Zhang et al., 2026; Li et al., 2026).

Router parameterization. At site j, let $h_{i,j,t} \in \mathbb{R}^D$ be the hidden representation of valid token t, and stack these representations as $\mathbf{H}_{i,j} = [h_{i,j,t}^\top]_{t=1}^{n_i} \in \mathbb{R}^{n_i \times D}$. We take their mean and elementwise maximum, normalize the two pooled vectors separately, and concatenate them:

$$\phi_{i,j} = \left[N_j^{\text{mean}}\left(\frac{1}{n_i}\sum_{t=1}^{n_i} h_{i,j,t}\right) ; N_j^{\max}\left(\max_{1 \le t \le n_i} h_{i,j,t}\right) \right],$$
$$\mathbf{p}_{i,j} = \text{softmax}(A_j \phi_{i,j} + b_j) = \left(p_{i,j}^{\text{E}}, p_{i,j}^{\text{C}}\right). \tag{1}$$

Here N_j^{mean} and $N_j^{\max}$ are separate learned RMS normalizers; $[\cdot\,;\cdot]$ denotes concatenation. Each site's normalizers and projection (A_j, b_j) are independent, with $A_j \in \mathbb{R}^{2 \times 2D}$. The mean summarizes the chunk, while the maximum retains strong responses at individual positions.

Hard execution with straight-through gradients. The forward pass chooses a deterministic binary action; training uses a straight-through gradient estimator (Bengio et al., 2013):

$$\widehat{\mathbf{a}}_{i,j} = \text{onehot}\left(\arg\max_{c\in\{\text{E,C}\}} p^{c}_{i,j}\right), \qquad \widetilde{\mathbf{a}}_{i,j} = \mathbf{p}_{i,j} + \text{sg}(\widehat{\mathbf{a}}_{i,j} - \mathbf{p}_{i,j}), \tag{2}$$

Here sg denotes stop-gradient, so $\widetilde{\mathbf{a}}_{i,j}$ has the forward value of $\widehat{\mathbf{a}}_{i,j}$ and the derivative of $\mathbf{p}_{i,j}$.

A site is reached only if every earlier site continued. Treating the hard topology as stop-gradient, define

$$\widehat{s}_{i,1} = 1, \qquad \widehat{e}_{i,j} = \widehat{s}_{i,j}\widehat{a}^{\text{E}}_{i,j}, \qquad \widehat{s}_{i,j+1} = \widehat{s}_{i,j}\widehat{a}^{\text{C}}_{i,j}. \tag{3}$$

Here $\widehat{s}_{i,j}$ marks that chunk i reaches site j and $\widehat{e}_{i,j}$ that its first exit occurs there; at most one $\widehat{e}_{i,j}$ is nonzero, and if none is, $\widehat{s}_{i,K+1}{=}1$ and the chunk reaches full depth. Hats denote hard, non-differentiated execution variables; tildes denote hard values carrying a soft surrogate gradient.

For an eligible chunk, the two depths are

$$D_i = d_1 + \sum_{j=1}^{K} |\mathcal{G}_j|\widehat{s}_{i,j+1},$$

$$W_i = |\mathcal{I}_{\text{state}} \cap [0, d_1)| + \sum_{j=1}^{K} |\mathcal{I}_{\text{state}} \cap \mathcal{G}_j|\widehat{s}_{i,j+1}. \tag{4}$$

All LaCT blocks are stateful, so $D_i = W_i$. For TTT-E2E suffix routing, $D_i = 18 + W_i$. Reported average depth means execution depth D_i unless stated otherwise.

Routing the persistent state. Let $\mathcal{S}_{i-1,\ell}$ be the persistent state of stateful layer $\ell \in \mathcal{I}_{\text{state}}$ before chunk i, including optimizer state when present. On an eligible chunk, each stateful layer in the unconditional stem or a continued group commits the candidate state computed on the routed trajectory. Every skipped stateful layer keeps its incoming state exactly, $\mathcal{S}_{i,\ell} = \mathcal{S}_{i-1,\ell}$ (formal statement in Appendix A). An exit suppresses remaining writes without undoing those already committed. The terminal prediction chunk runs at full depth without updating persistent state.

The hard variable $\widehat{s}_{i,j+1}$ determines whether $\mathcal{G}_j$ executes. For stateful layers in that group, we use $\gamma_{i,j} = \widehat{s}_{i,j+1}\widetilde{a}^{\text{C}}_{i,j}$ to differentiate the state update. TTT-E2E applies this factor to the affected inner gradients; LaCT uses it to select between incoming and candidate fast-weight and momentum states. State-transition gradients therefore propagate only through groups executed in the hard forward pass.

For skipped layers, we compute position-aligned K/V from the detached representation at the exit boundary. Their fast weights and optimizer states remain unchanged.

3.3 Exit Adapters and Two-Stage Readout Alignment

Intermediate representations need not match the input expected by the frozen LM head. We align them using learned readouts (Belrose et al., 2025) and intermediate-exit supervision (Chen et al., 2024; Elhoushi et al., 2024). A poor shallow prediction may reflect an untrained readout rather than insufficient depth, so we prepare the readouts before enabling routing. Each candidate depth has a low-rank residual adapter and an independent normalization:

$$R_j(h) = N^{\text{exit}}_j(h + U_j\,\text{SiLU}(V_j h)), \qquad V_j \in \mathbb{R}^{r\times D}, \quad U_j \in \mathbb{R}^{D\times r}. \tag{5}$$

Applied tokenwise, R_j produces the normalized exit representation $\mathbf{R}_{i,j} = [R_j(h_{i,j,t})^\top]_{t=1}^{n_i}$, decoded by the shared frozen head W_{LM}. We set $U_j{=}0$ and copy the final RMSNorm scale into N^{exit}_j, initializing an identity residual at the pretrained readout scale.

Training proceeds in two stages (Figure 8, Appendix B). In *Stage 1*, we disable routing and train only the exit readouts on detached boundary representations. All eligible chunks execute every layer and commit all native updates. *Stage 2* enables hard routing and jointly trains the routers and readouts, while retaining auxiliary readout supervision. The backbone, final normalization, and LM head remain frozen.

3.4 GATE-TEACHER COUNTERFACTUAL

An exit gate learns by comparing its selected readout with a continuation. A continuation from the student's memory already reflects earlier routing decisions. We therefore use a detached teacher with its own full-depth update history to provide the gate's training reference.

Stage 2 evaluates three trajectories. The hard-policy *student* produces the predictions and maintains the committed fast states. A *continuation stream* matches it until exit, then a detached ghost evaluates the omitted groups without committing updates. It supplies the normalized full-depth representation $\mathbf{H}_i^{\text{cont}}$ and the deeper probabilities used by soft survival. A detached *teacher* starts from the same initialization and follows its own full-depth history, committing every update. Neither auxiliary trajectory copies its states into the student. GTCF replaces the continuation used in the selected exit gate's straight-through derivative with the teacher representation (Figure 7, Appendix A).

In Stage 2, for an eligible chunk, use the stopped first-exit selector $\widehat{e}_{i,j}$ from equation 3 to define

$$\widehat{\alpha}_i = \sum_{j=1}^{K} \widehat{e}_{i,j}, \qquad \widetilde{\alpha}_i = \sum_{j=1}^{K} \widehat{e}_{i,j}\widetilde{a}_{i,j}^{\text{E}}, \qquad \mathbf{R}_i^{\text{E}} = \sum_{j=1}^{K} \widehat{e}_{i,j}\mathbf{R}_{i,j}. \tag{6}$$

$\widehat{\alpha}_i$ records an exit, $\widetilde{\alpha}_i$ has the same value but carries the selected gate's derivative, and $\mathbf{R}_i^{\text{E}}$ is its readout; all are zero at full depth.

Without GTCF, the task readout is $\widetilde{\alpha}_i\mathbf{R}_i^{\text{E}} + (1-\widetilde{\alpha}_i)\mathbf{H}_i^{\text{cont}}$. GTCF substitutes the detached teacher only in the counterfactual branch whose coefficient is zero in the hard forward pass. Let $\mathbf{T}_i$ be the teacher's normalized full-depth representation and define

$$\mathbf{H}_i^{\text{cf}} = \widehat{\alpha}_i\,\text{sg}(\mathbf{T}_i) + (1-\widehat{\alpha}_i)\mathbf{H}_i^{\text{cont}}, \qquad \widetilde{\mathbf{H}}_i = \widetilde{\alpha}_i\mathbf{R}_i^{\text{E}} + (1-\widetilde{\alpha}_i)\mathbf{H}_i^{\text{cf}}. \tag{7}$$

At an exit, $\widehat{\alpha}_i = \widetilde{\alpha}_i = 1$ in value and $\widetilde{\mathbf{H}}_i = \mathbf{R}_i^{\text{E}}$; otherwise the ordinary full-depth value remains. For fixed incoming student state and hard routing decisions, the substitution preserves the current chunk's logits and cross-entropy value. The gate comparator changes from $\mathbf{R}_i^{\text{E}} - \mathbf{H}_i^{\text{cont}}$ to $\mathbf{R}_i^{\text{E}} - \text{sg}(\mathbf{T}_i)$; no gradient reaches the teacher, and the exit readout retains its LM gradient.

In TTT-E2E, the routed cross-entropy also defines the inner fast-weight update, so the surrogate can influence earlier fast-weight layers through the gate's input representations. LaCT's local key–value update is separate from this language-model comparator. The teacher is used during training and omitted at inference (TTT-E2E's inference comparator is specified in Appendix A.1). It supplies a full-depth reference for credit assignment, in the spirit of counterfactual baselines for discrete decisions (Foerster et al., 2024), but follows a different fast-state history from the student.

3.5 TRAINING OBJECTIVE

For minibatch $\mathcal{B}$, let $\mathcal{M}_b$ contain sequence b's eligible chunks. $\mathcal{L}_{\text{LM}}$ is the next-token cross-entropy of $\widetilde{\mathbf{H}}_{b,i}$, averaged over valid tokens. $\mathcal{L}_{\text{exit}}$ applies the same supervision at each exit reached by the hard route, using detached boundary representations and reachability $\widehat{s}_{b,i,j}$. It therefore trains only readouts, not upstream representations or reach decisions (definitions: Appendix A).

Efficiency loss. Language-model supervision does not specify a compute budget, so we penalize continued execution using a normalized cost proxy (Graves, 2017; Banino et al., 2021). At each reached site, EXIT adds no routed-group cost, whereas CONTINUE incurs the cost c_{grp} of executing the complete next group. Let c_{pre} denote computation before the first routing site. Reaching a deeper group requires earlier gates to continue, so Stage 2 weights its cost by soft survival: $\rho_{b,i,1} = 1$ and $\rho_{b,i,j} = \prod_{k<j} p_{b,i,k}^{\text{C}}$; we use batches with $|\mathcal{M}_b| > 0$.

$$C_{b,i}^{\text{soft}} = \frac{c_{\text{pre}} + c_{\text{grp}}\sum_{j=1}^{K}\rho_{b,i,j}p_{b,i,j}^{\text{C}}}{c_{\text{pre}} + Kc_{\text{grp}}},$$
$$\mathcal{L}_{\text{eff}} = \frac{1}{|\mathcal{B}|}\sum_{b\in\mathcal{B}}\frac{1}{|\mathcal{M}_b|}\sum_{i\in\mathcal{M}_b} C_{b,i}^{\text{soft}}. \tag{8}$$

The product makes each site's cost differentiable; averaging within sequences gives each sequence equal weight. Stage 2 evaluates every router head; at sites the hard path does not reach, probabilities

come from the detached ghost and affect neither forward value nor persistent state. Ghost continuation is unnecessary at inference; FLOP accounting includes the router and readout computations retained by the implementation (§4.3).

Router z-loss. To control router-logit growth, we use the z-loss $\mathcal{L}_z$ of Zoph et al. (2022), which regularizes the logits' log-partition value (definition in Appendix A).

We write the outer objective as

$$\mathcal{L} = \mathcal{L}_{\text{LM}} + \lambda_{\text{exit}}\mathcal{L}_{\text{exit}} + \lambda_{\text{eff}}\mathcal{L}_{\text{eff}} + \lambda_z\mathcal{L}_z. \tag{9}$$

Stage 1 optimizes only $\mathcal{L}_{\text{exit}}$. In Stage 2, $\mathcal{L}_{\text{eff}}$ reaches every router through soft survival; $\mathcal{L}_{\text{LM}}$ trains the selected straight-through gate and backpropagates through state operations only on the hard route. GTCF changes the differentiation rule at the selected exit gate rather than adding a scalar loss term; in TTT-E2E, this can also affect inner fast-weight gradients.

4 Experiments

4.1 Setup

Models. LaCT is our primary testbed for AdaWrite; TTT-E2E tests applicability to a second update mechanism. Both are frozen 1B, 24-layer backbones (routing settings: Appendix B). We pretrain the LaCT backbone from scratch on ProLong-64K (Gao et al., 2025) at a 65,536-token context, following LaCT's architecture and optimization design (Zhang et al., 2026); all 24 layers contain fast weights, and the frozen checkpoint's ordinary full-depth execution is the LaCT baseline. For TTT-E2E we use the released checkpoint unchanged (Tandon et al., 2025); only its last six layers contain fast weights. We place five LaCT sites at depths $\{4, 8, 12, 16, 20\}$ and three TTT-E2E sites at $\{18, 20, 22\}$, with rank-128 readouts (2.0M and 1.6M trainable parameters, approximately 0.18% and 0.16% of the respective backbones). Each routing patch is trained on the same long-context source distribution used for its frozen backbone (ProLong-64K for LaCT and Books3 (Gao et al., 2020) for TTT-E2E), so it introduces no task-specific or evaluation-specific data. We use one hard policy per backbone with no per-task tuning.

Evaluation. The learned policies route only complete prefill chunks. Routed models, full-depth baselines, and fixed-depth controls all run the terminal prefill chunk and scored or generated target tokens at full depth; the terminal chunk commits no fast-weight update and is excluded from *average executed depth*, the mean over eligible prefill chunks. S-NIAH-1 is our primary controlled retrieval test, following the small-model presentation in LaCT (Zhang et al., 2026); the remaining tasks characterize the same policy on additional inputs. (i) **RULER S-NIAH-1/2/3** (Hsieh et al., 2024): needle retrieval in repetitive filler (S-NIAH-1) or essays (S-NIAH-2/3), with 100 samples per task per length at $\{8, 16, 32, 64, 96, 128\}$K tokens; lengths beyond each backbone's training context (64K for LaCT, 8K for TTT-E2E) probe length extrapolation; we report accuracy averaged over the six lengths. (ii) **LongBench v2** (Bai et al., 2025): the 358-example short/medium subset (longer examples exceed available GPU memory), selecting the A/B/C/D answer label with the highest summed conditional log-likelihood and macro-averaging accuracy over 17 subtasks. (iii) **PG-19** (Rae et al., 2019): 39 held-out books, each with a routed 65,536-token context followed by a 256-token full-depth continuation; we pool perplexity over these 9,984 tokens.

4.2 Main Results

Table 1 compares prediction quality and executed depth for AdaWrite and each backbone's full-depth baseline.

(1) Routing improves LaCT's long-range retrieval extrapolation. On S-NIAH-1, the routed model executes 8.2–13.8 of 24 layers. Accuracy is 0.06–0.09 lower through 32K, comparable at 64K (0.90 vs. 0.89), and much stronger beyond the 64K pretraining length: 0.79 vs. 0.50 at 96K and 0.40 vs. 0.00 at 128K. Mean accuracy rises from 0.73 to 0.81 (Figure 2). Answers use the same full-depth frozen readout, so the improvement comes from routing during context processing. The policy trades shorter-context accuracy for stronger extrapolation at lower analytical compute.

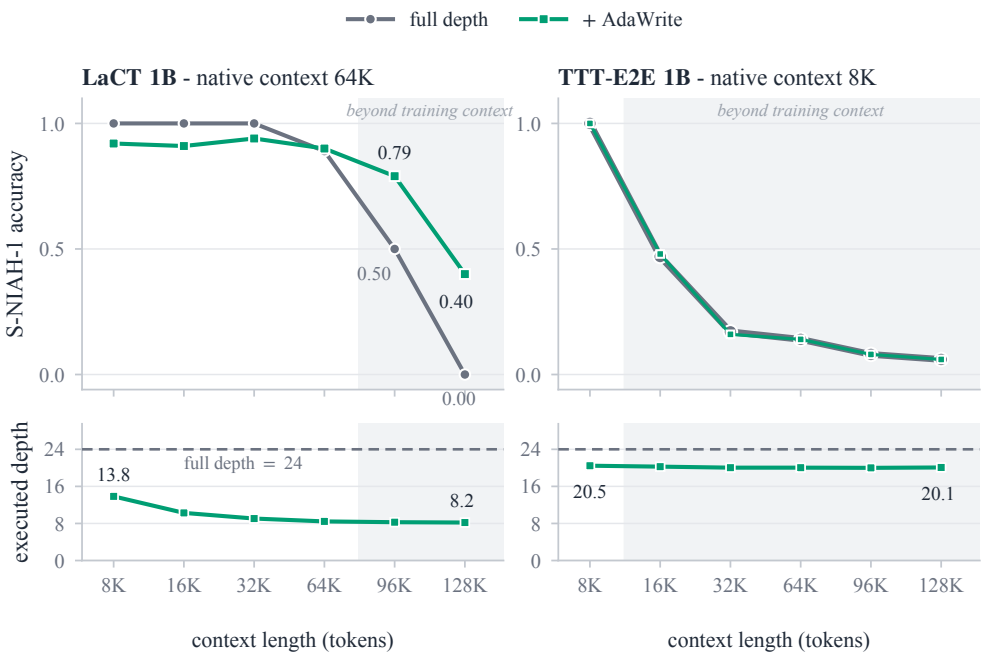


Figure 2: **S-NIAH-1 accuracy and executed depth by context length** (shading: beyond the training context; Appendix D). LaCT routing lowers accuracy through 32K, is comparable at 64K, and improves retrieval at 96K/128K; TTT-E2E tracks its baseline.

Table 1: Main results. S-NIAH accuracy is averaged over six lengths (Appendix D); LongBench v2 uses macro accuracy over 17 subtasks; PG-19 pools perplexity over each book's 256-token continuation after a 65,536-token context. "D" denotes mean executed depth over eligible prefill chunks. TTT-E2E always executes its 18-layer prefix. Bold marks the numerically better result in each pair.

	S-NIAH-1 ↑		S-NIAH-2 ↑		S-NIAH-3 ↑		LongBench v2 ↑		PG-19 ppl ↓	
Model	Acc	D	Acc	D	Acc	D	Acc	D	Ppl	D
LaCT 1B (full)	0.73	24.0	0.148	24.0	0.153	24.0	0.260	24.0	**16.82**	24.0
+ AdaWrite	**0.81**	9.7	**0.157**	22.5	**0.157**	22.8	**0.272**	22.4	17.11	22.1
TTT-E2E 1B (full)	0.32	24.0	**0.230**	24.0	**0.208**	24.0	**0.249**	24.0	**11.84**	24.0
+ AdaWrite	0.32	20.1	0.228	23.4	0.203	23.3	0.237	23.7	12.24	22.4

(2) One policy adapts write depth across inputs. LaCT uses 8 to 14 layers on S-NIAH-1 and retains 22 to 23 on natural text and S-NIAH-2/3. It scores 0.272 vs. 0.260 on LongBench v2 at depth 22.4, and 17.11 vs. 16.82 perplexity on PG-19 at depth 22.1. S-NIAH-2/3 remain challenging for the full-depth backbone beyond 16K; routing stays close to its accuracy while retaining most layers. On S-NIAH-1, deep routes concentrate near the needle (§4.4). These depth differences arise without task-specific routing rules.

(3) The procedure also applies to TTT-E2E. Routing TTT-E2E's last six layers matches average S-NIAH-1 accuracy (0.32) at execution depth 20.1, or write depth 2.1 among six stateful layers. Other task differences are small: PG-19 perplexity rises from 11.84 to 12.24 while analytical FLOPs fall by a factor of 1.13. Appendix G compares suffix and whole-network routing.

4.3 Inference FLOPs

We estimate analytical prefill FLOPs from component formulas and observed routes on the samples in Table 1, using $2mnk$ per matrix multiplication. Counts cover forward computation and

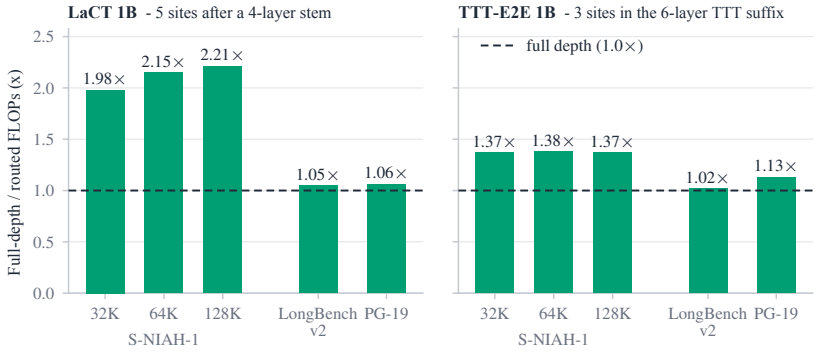


Figure 3: **Analytical prefill FLOP ratio** (full depth / AdaWrite), from observed routes. Savings are largest on S-NIAH-1 and smaller on natural text, where routing uses more layers. Both panels include whole-model computation. Appendix C reports GFLOPs per processed token.

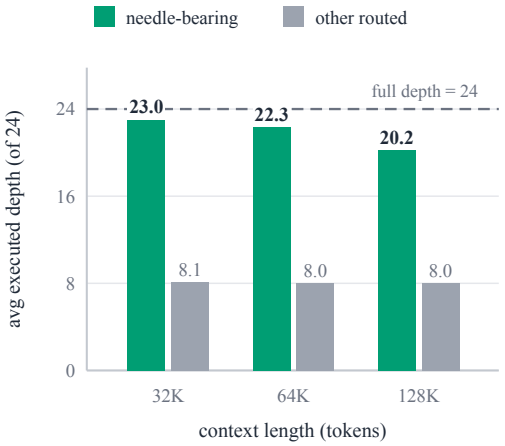


Figure 4: **Where the depth goes** (AdaWrite on LaCT 1B, S-NIAH-1). Mean depth of needle-bearing and other routed chunks, excluding the unrouted terminal chunk (Appendix E).

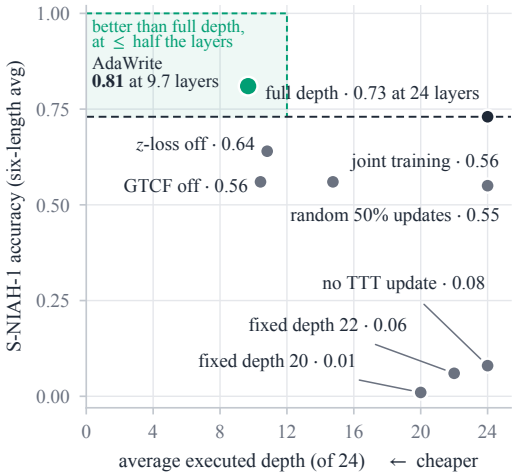


Figure 5: **AdaWrite ablations** (Table 4): S-NIAH-1 accuracy vs. average executed depth, LaCT 1B.

fast-weight updates; for PG-19 they also include the teacher-forced continuation. Autoregressive generation is excluded for both models. Figure 3 reports the full-depth to routed ratios; Appendix C gives GFLOPs per processed token and accounting assumptions.

On S-NIAH-1 at 32K, 64K, and 128K, LaCT reduces analytical FLOPs by factors of 1.98 to 2.21, versus 1.05 to 1.06 on natural text. Routing TTT-E2E's six-layer suffix reduces whole-model FLOPs by factors of 1.37 to 1.38 on S-NIAH-1, 1.02 on LongBench v2, and 1.13 on PG-19. These are analytical savings, not measured latency improvements.

4.4 Where the Depth Goes: Needle-Bearing Chunk Analysis

The router receives no RULER or needle supervision. For analysis, we label routed 2,048-token chunks that overlap the needle in 100 S-NIAH-1 prompts at each of 32K, 64K, and 128K. These chunks average 23.0/22.3/20.2 layers, versus 8.1/8.0/8.0 for other chunks. They make up only 6.6/3.1/1.6% of routed chunks but account for 92.1/97.7/95.0% of full-depth routes. Every other full-depth chunk immediately follows a needle-bearing one: the learned policy concentrates deep writes near the evidence. Appendix E gives the protocol and per-length statistics.

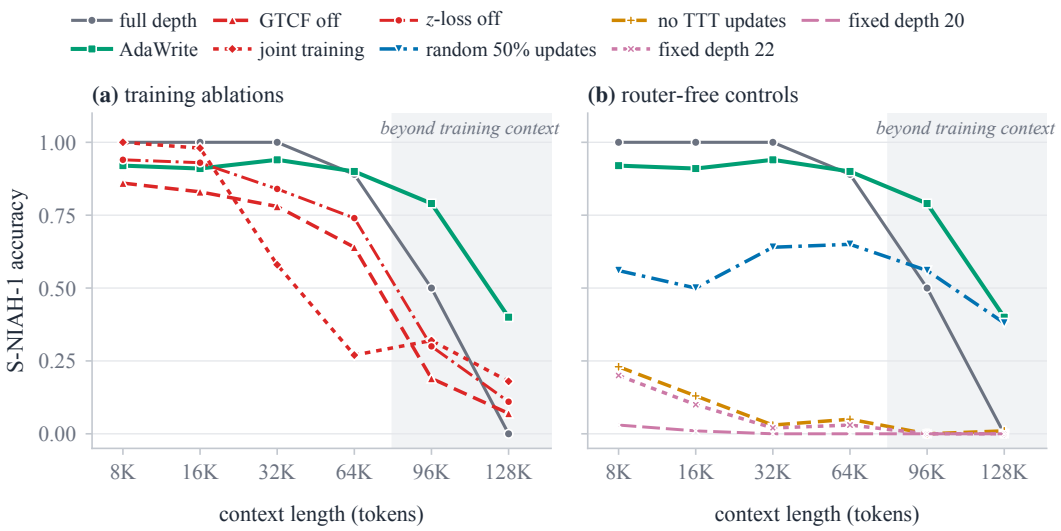


Figure 6: **AdaWrite ablations by context length** (LaCT 1B, S-NIAH-1; Table 5). (a) GTCF, readout warm-up, and the z-loss help most at longer contexts. (b) Random 50% updates improve extrapolation but reduce shorter-context accuracy; fixed depths and no updates perform worse.

5 Ablation Studies

We compare training components, fixed depths, and update schedules on LaCT (Figure 5, Figure 6); Table 4 and Appendix F give complete results and discussion.

GTCF. Using the ghost instead of the teacher lowers S-NIAH-1 accuracy from 0.81 to 0.56 at similar depth (10.4 vs. 9.7), supporting the independent teacher comparator (Appendix F, GTCF).

Two-stage readout alignment. Joint training without the readout-only stage gives 0.56 accuracy at 14.8 layers, versus 0.81 at 9.7 after readout alignment (Appendix F, readout alignment).

Router z-loss. Removing the z-loss lowers S-NIAH-1 accuracy to 0.64 at depth 10.8 and raises PG-19 perplexity to 18.28 at depth 18.8 (Appendix F, z-loss).

Fixed-depth controls. Fixed prefill depths 20 and 22 give 0.01 and 0.06 accuracy with the same readout rank and budget, despite greater depth than learned routing (Appendix F, fixed depth).

Update-schedule controls. Removing updates gives 0.08 accuracy; independently retaining each layer–chunk update with probability 0.5 gives 0.55, both at full depth. Random thinning improves extrapolation but reduces shorter-context accuracy; both controls outperform routing on LongBench v2 (0.305–0.308 vs. 0.272; Appendix F, update schedules).

Random thinning shows that dense writes are not uniformly preferable on S-NIAH-1. Routing reaches similar accuracy at 128K, higher accuracy at 96K, and a higher average across lengths, while using fewer layers.

6 Conclusion

AdaWrite reduces prefill computation in frozen TTT models by learning each chunk's execution and write depth. On LaCT S-NIAH-1, deep writes concentrate near the needle and retrieval improves beyond the training context at lower analytical FLOPs. The same policy uses more layers on other inputs, with quality close to its full-depth baseline. Ablations support the training components, and results on TTT-E2E show that the procedure also applies to a second update mechanism.

The Use of Large Language Models

We used large language models to edit and polish the writing, to assist in writing and reviewing code, to generate figures from author-provided results, to search and summarize related literature, and to assist in analyzing and interpreting results that the authors computed. We did not use them for synthetic data generation, theoretical modeling or proofs, methodology design, or as a source of experimental results. All research ideas, experiments, and claims were designed, executed, and verified by the authors, who take full responsibility for the content.

Reproducibility Statement

Appendix B specifies the provenance of both backbones and every routing-patch hyperparameter, initialization, and schedule. §4.1 defines the evaluation protocol, Appendix D reports per-length results, Appendix C gives the FLOP accounting, and Appendix E the needle-labeling protocol.

References

Joshua Ainslie, Tao Lei, Michiel de Jong, Santiago Ontañón, Siddhartha Brahma, Yury Zemlyanskiy, David Uthus, Mandy Guo, James Lee-Thorp, Yi Tay, Yun-Hsuan Sung, and Sumit Sanghai. Colt5: Faster long-range transformers with conditional computation, 2023. URL https://arxiv.org/abs/2303.09752.

Jimmy Ba, Geoffrey Hinton, Volodymyr Mnih, Joel Z. Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past, 2016. URL https://arxiv.org/abs/1610.06258.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation, 2025. URL https://arxiv.org/abs/2507.10524.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks, 2025. URL https://arxiv.org/abs/2412.15204.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder, 2021. URL https://arxiv.org/abs/2107.05407.

Ali Behrouz, Zeman Li, Praneeth Kacham, Majid Daliri, Yuan Deng, Peilin Zhong, Meisam Razaviyayn, and Vahab Mirrokni. Atlas: Learning to optimally memorize the context at test time, 2025a. URL https://arxiv.org/abs/2505.23735.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=8GjSf9Rh7Z.

Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. It's all connected: A journey through test-time memorization, attentional bias, retention, and online optimization. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=gZyEJ2kMow.

Nora Belrose, Igor Ostrovsky, Lev McKinney, Zach Furman, Logan Smith, Danny Halawi, Stella Biderman, and Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens, 2025. URL https://arxiv.org/abs/2303.08112.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL https://arxiv.org/abs/1308.3432.

Víctor Campos, Brendan Jou, Xavier Giró i Nieto, Jordi Torres, and Shih-Fu Chang. Skip RNN: Learning to skip state updates in recurrent neural networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HkwVAXyCW.

Yanxi Chen, Xuchen Pan, Yaliang Li, Bolin Ding, and Jingren Zhou. Ee-llm: Large-scale training and inference of early-exit large language models with 3d parallelism, 2024. URL https://arxiv.org/abs/2312.04916.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=S1di0sfgl.

Maha Elbayad, Jiatao Gu, Edouard Grave, and Michael Auli. Depth-adaptive transformer, 2020. URL https://arxiv.org/abs/1910.10073.

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, Ahmed Aly, Beidi Chen, and Carole-Jean Wu. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.681. URL http://dx.doi.org/10.18653/v1/2024.acl-long.681.

Guhao Feng, Shengjie Luo, Kai Hua, Ge Zhang, Di He, Wenhao Huang, and Tianle Cai. In-place test-time training, 2026. URL https://arxiv.org/abs/2604.06169.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients, 2024. URL https://arxiv.org/abs/1705.08926.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The pile: An 800gb dataset of diverse text for language modeling, 2020. URL https://arxiv.org/abs/2101.00027.

Tianyu Gao, Alexander Wettig, Howard Yen, and Danqi Chen. How to train long-context language models (effectively), 2025. URL https://arxiv.org/abs/2410.02660.

Alex Graves. Adaptive computation time for recurrent neural networks, 2017. URL https://arxiv.org/abs/1603.08983.

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

Shwai He, Tao Ge, Guoheng Sun, Bowei Tian, Xiaoyang Wang, and Dong Yu. Router-tuning: A simple and effective approach for enabling dynamic-depth in transformers, 2025. URL https://arxiv.org/abs/2410.13184.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models?, 2024. URL https://arxiv.org/abs/2404.06654.

Filipe Laitenberger, Dawid Jan Kopiczko, Cees G. M. Snoek, and Yuki M Asano. What layers when: Learning to skip compute in LLMs with residual gates. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=aiP6XfaYZR.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alexandros G. Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar.

Datacomp-lm: In search of the next generation of training sets for language models, 2025. URL https://arxiv.org/abs/2406.11794.

Zeman Li, Ali Behrouz, Yuan Deng, Peilin Zhong, Praneeth Kacham, Mahdi Karami, Meisam Razaviyayn, and Vahab Mirrokni. TNT: Improving chunkwise training for test-time memorization. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=rajioNWfRs.

Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, Zhiqi Huang, Huan Yuan, Suting Xu, Xinran Xu, Guokun Lai, Yanru Chen, Huabin Zheng, Junjie Yan, Jianlin Su, Yuxin Wu, Yutao Zhang, Zhilin Yang, Xinyu Zhou, Mingxing Zhang, and Jiezhong Qiu. MoBA: Mixture of block attention for long-context LLMs. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=RlqYCpTu1P.

Xuan Luo, Weizhi Wang, and Xifeng Yan. Adaptive layer-skipping in pre-trained llms, 2025. URL https://arxiv.org/abs/2503.23798.

Ziqiao Ma, Xueyang Yu, Haoyu Zhen, Yuncong Yang, Joyce Chai, and Chuang Gan. Fast spatial memory with elastic test-time training, 2026. URL https://arxiv.org/abs/2604.07350.

Lingrui Mei, Shenghua Liu, Yiwei Wang, Yuyao Ge, Baolong Bi, Jiayu Yao, Jun Wan, Ziling Yin, Jiafeng Guo, and Xueqi Cheng. Gated differentiable working memory for long-context language modeling, 2026. URL https://arxiv.org/abs/2601.12906.

Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yaofo Chen, Shijian Zheng, Peilin Zhao, and Mingkui Tan. Efficient test-time model adaptation without forgetting, 2022. URL https://arxiv.org/abs/2204.02610.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling, 2019. URL https://arxiv.org/abs/1911.05507.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models, 2024. URL https://arxiv.org/abs/2404.02258.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers, 2021. URL https://arxiv.org/abs/2102.11174.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. Confident adaptive language modeling, 2022. URL https://arxiv.org/abs/2207.07061.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017. URL https://arxiv.org/abs/1701.06538.

Sainbayar Sukhbaatar, Da Ju, Spencer Poff, Stephen Roller, Arthur Szlam, Jason Weston, and Angela Fan. Not all memories are created equal: Learning to forget by expiring, 2021. URL https://arxiv.org/abs/2105.06548.

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts, 2020. URL https://arxiv.org/abs/1909.13231.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): Rnns with expressive hidden states, 2025. URL https://arxiv.org/abs/2407.04620.

Arnuv Tandon, Karan Dalal, Xinhao Li, Daniel Koceja, Marcel Rød, Sam Buchanan, Xiaolong Wang, Jure Leskovec, Sanmi Koyejo, Tatsunori Hashimoto, Carlos Guestrin, Jed McCaleb, Yejin Choi, and Yu Sun. End-to-end test-time training for long context, 2025. URL https://arxiv.org/abs/2512.23675.

Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization, 2021. URL https://arxiv.org/abs/2006.10726.

Zixuan Wang, Xingyu Dang, Rui-Jie Zhu, Zixin Wen, Hengyu Fu, Wenhao Chai, and Jason D. Lee. Learning what to remember: Test-time training via context distillation, 2026. URL https://arxiv.org/abs/2608.01672.

Juyun Wee, Minjae Park, and Jaeho Lee. Prompt-based depth pruning of large language models. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=hRxHF1xPYB.

Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers, 2022. URL https://arxiv.org/abs/2203.08913.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. Deebert: Dynamic early exiting for accelerating bert inference, 2020. URL https://arxiv.org/abs/2004.12993.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training, 2024. URL https://arxiv.org/abs/2312.06635.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule, 2025. URL https://arxiv.org/abs/2412.06464.

yikun jiang, Huanyu Wang, Lei Xie, Hanbin Zhao, Chao Zhang, Hui Qian, and John C.S. Lui. D-LLM: A token adaptive computing resource allocation strategy for large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=UIOjGTKHQG.

Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. Native sparse attention: Hardware-aligned and natively trainable sparse attention, 2025. URL https://arxiv.org/abs/2502.11089.

Tianyuan Zhang, Sai Bi, Yicong Hong, Kai Zhang, Fujun Luan, Songlin Yang, Kalyan Sunkavalli, William T. Freeman, and Hao Tan. Test-time training done right. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=Tb9qAxT3xv.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. St-moe: Designing stable and transferable sparse expert models, 2022. URL https://arxiv.org/abs/2202.08906.

A Backbone Update Rules and Router Regularization

For completeness, we give the native chunk transitions of the two backbones in the notation of §3.1, and the router z-loss of §3.5.

LaCT. LaCT uses a fast network f_Ω as an associative memory (Zhang et al., 2026). For one stateful layer, suppressing its layer index, let the query, key, and value projections be (q_t, k_t, v_t) and the predicted learning rates be η_t. It aggregates a local key–value association gradient with $\ell_{\mathrm{KV}}(f_\Omega(k), v) = -f_\Omega(k)^\top v$:

$$G_i = \nabla_\Omega \sum_{t \in \mathcal{C}_i} \eta_t \, \ell_{\mathrm{KV}}(f_\Omega(k_t), v_t)\Big|_{\Omega=\Omega_{i-1}},$$
$$\mathcal{S}_i = \mathrm{Update}(\mathcal{S}_{i-1}, G_i; \mathcal{C}_i), \qquad o_t = f_{\Omega_{i-1}}(q_t), \quad t \in \mathcal{C}_i. \tag{10}$$

Here Ω_{i-1} is the fast-weight component of $\mathcal{S}_{i-1}$ and Update is LaCT's native stateful optimizer, including chunk-conditioned coefficients such as predicted momentum; the incoming memory answers current queries before the update commits, and the inner objective is local to each layer rather than the LM loss. An executed LaCT layer retains this local objective, optimizer, and read-before-write schedule; routing suppresses the updates of skipped layers.

TTT-E2E. TTT-E2E continues next-token training on the test sequence itself (Tandon et al., 2025). Let $\ell_t(\Omega) = \mathrm{CE}(f_\theta(x_{\le t-1}; \Omega), x_t)$. The full-depth backbone's chunk update and meta-training objective are

$$\Omega_i = \Omega_{i-1} - \frac{\eta}{n_i} \sum_{t \in \mathcal{C}_i} \nabla_\Omega \ell_t(\Omega_{i-1}), \qquad \mathcal{L}_{\mathrm{E2E}}(\Omega_0; \mathbf{x}) = \frac{1}{T} \sum_{i=1}^{M} \sum_{t \in \mathcal{C}_i} \ell_t(\Omega_{i-1}). \tag{11}$$

Meta-training differentiates through these updates to learn an initialization Ω_0 whose updates improve predictions as context accumulates; in our backbone, MLPs in a deep suffix supply the fast weights and sliding-window attention provides token mixing. Under routing, the cross-entropy is evaluated through the routed predictor and differentiated with its straight-through gates; gradients are applied only to executed fast-weight layers. During Stage 2, this differentiation includes GTCF (§3.4). At inference, the inner update uses the teacher-free routed predictor, retaining straight-through gate derivatives (Appendix A.1).

Router z-loss. To stabilize router training, we regularize the router logits $\mathbf{z}_{b,i,j} = A_j \phi_{b,i,j} + b_j$ of equation 1 with a z-loss (Zoph et al., 2022),

$$\mathcal{L}_z = \frac{\sum_{b \in \mathcal{B}} \sum_{i \in \mathcal{M}_b} \sum_{j=1}^{K} \widehat{s}_{b,i,j} \Big(\log \sum_{c \in \{\mathrm{E},\mathrm{C}\}} \exp\left(z^c_{b,i,j}\right) \Big)^2}{\max\left\{1, \sum_{b \in \mathcal{B}} \sum_{i \in \mathcal{M}_b} \sum_{j=1}^{K} \widehat{s}_{b,i,j}\right\}}. \tag{12}$$

Only sites reached by the stopped hard policy contribute. The penalty acts on the log-partition value of their logits.

Routed state transition. For an eligible chunk, let $\mathcal{S}_{i-1,\ell}$ be the persistent state of a stateful layer $\ell \in \mathcal{I}_{\mathrm{state}}$ before chunk i, including optimizer state when present. Let $\mathcal{S}^{\mathrm{cand}}_{i,\ell}$ be the candidate state produced by the routed update rule when that layer executes. The shared hard-forward invariant, defined only for stateful layers, is

$$\mathcal{S}_{i,\ell} = \begin{cases} \mathcal{S}^{\mathrm{cand}}_{i,\ell}, & \ell \in \mathcal{I}_{\mathrm{state}} \cap [0, d_1), \\ \mathcal{S}^{\mathrm{cand}}_{i,\ell}, & \ell \in \mathcal{I}_{\mathrm{state}} \cap \mathcal{G}_j \text{ and } \widehat{s}_{i,j+1} = 1, \\ \mathcal{S}_{i-1,\ell}, & \ell \in \mathcal{I}_{\mathrm{state}} \cap \mathcal{G}_j \text{ and } \widehat{s}_{i,j+1} = 0. \end{cases} \tag{13}$$

Language-model and exit-readout losses. For a minibatch $\mathcal{B}$, index sequences by b and let $\mathcal{M}_b$ be the eligible chunks of sequence b. Let $m_{b,i,t} \in \{0,1\}$ be the valid-token mask, $y_{b,i,t}$ the next-token target, and $\widetilde{h}_{b,i,t}$ the corresponding row of $\widetilde{\mathbf{H}}_{b,i}$. During Stage 1, and for $i \notin \mathcal{M}_b$ in Stage 2, this representation is the ordinary normalized full-depth output, and the hard reachability $\widehat{s}_{b,i,j}$ of equation 3 equals one for every site during Stage 1. With token losses $\ell^{\mathrm{LM}}_{b,i,t} = \mathrm{CE}(W_{\mathrm{LM}} \widetilde{h}_{b,i,t}, y_{b,i,t})$ and $\ell^{\mathrm{exit}}_{b,i,j,t}$ defined by

$$\ell^{\mathrm{exit}}_{b,i,j,t} = \mathrm{CE}(W_{\mathrm{LM}} R_j(\mathrm{sg}(h_{b,i,j,t})), y_{b,i,t}), \tag{14}$$

the aggregated losses are

$$\mathcal{L}_{\mathrm{LM}} = \frac{\sum_{b \in \mathcal{B}} \sum_{i,t} m_{b,i,t} \, \ell^{\mathrm{LM}}_{b,i,t}}{\sum_{b \in \mathcal{B}} \sum_{i,t} m_{b,i,t}},$$
$$\mathcal{L}_{\mathrm{exit}} = \frac{\sum_{b \in \mathcal{B}} \sum_{i \in \mathcal{M}_b} \sum_{j,t} \widehat{s}_{b,i,j} m_{b,i,t} \, \ell^{\mathrm{exit}}_{b,i,j,t}}{\sum_{b \in \mathcal{B}} \sum_{i \in \mathcal{M}_b} \sum_{j,t} \widehat{s}_{b,i,j} m_{b,i,t}}. \tag{15}$$

In Stage 2, each readout is supervised whenever it is reachable under the hard policy.

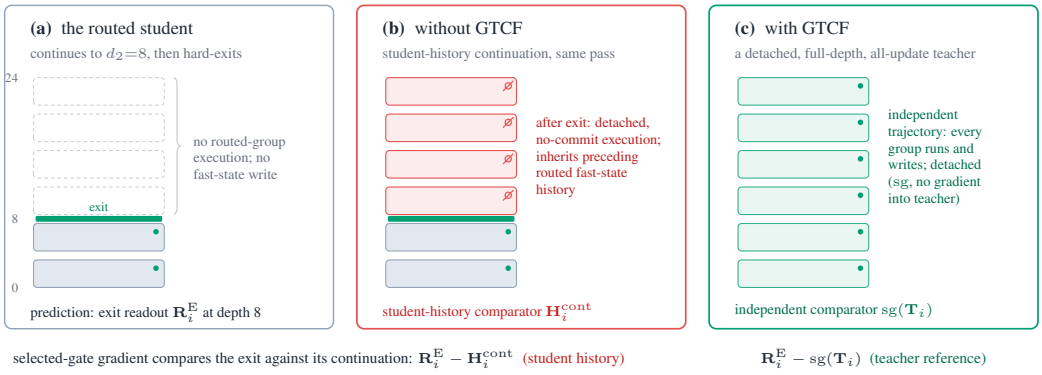


Figure 7: **Gate-Teacher Counterfactual in AdaWrite.** The gate compares its selected exit (a) with a continuation. Without GTCF (b), the continuation uses the student's routed fast states but commits no updates. GTCF (c) uses a detached teacher that follows its own full-depth state history and commits every update. For fixed incoming student state and hard decisions, the current chunk's logits and cross-entropy value are preserved. No gradient flows into the teacher; in TTT-E2E, the changed gate derivative can affect inner fast-weight updates.

A.1 TTT-E2E INFERENCE

Inference omits the teacher and ghost continuation. For an eligible chunk that exits at site j, let $\mathbf{H}_i^{\mathrm{E}} = \mathbf{H}_{i,j}$ be its boundary representation and let N_{final} apply the frozen final normalization token-wise. The teacher-free comparator and task readout are

$$\mathbf{C}_i^{\mathrm{infer}} = N_{\mathrm{final}}\big(\mathrm{sg}(\mathbf{H}_i^{\mathrm{E}})\big)\,, \qquad \widetilde{\mathbf{H}}_i^{\mathrm{infer}} = \widetilde{\alpha}_i \mathbf{R}_i^{\mathrm{E}} + (1-\widetilde{\alpha}_i)\mathbf{C}_i^{\mathrm{infer}}. \tag{16}$$

Here $\widetilde{\alpha}_i$ and $\mathbf{R}_i^{\mathrm{E}}$ are defined in equation 6. A chunk that does not exit uses its ordinary normalized full-depth output. Slow backbone parameters, routers, and readouts remain fixed; only fast weights adapt.

For each updating chunk, inference proceeds as follows:

1. Read the incoming fast weights and follow the hard route. After an exit, preserve skipped fast states and calculate their position-aligned K/V from the detached exit boundary.
2. Decode the task readout through the frozen LM head and form its cross-entropy $\mathcal{L}_i^{\mathrm{infer}}$, averaged over valid tokens. Differentiate this loss with respect to the incoming fast weights, retaining the selected gate's straight-through derivative through its input representations.
3. Mask gradients to executed fast-weight layers before clipping and applying the configured SGD step. With $\widehat{w}_{i,\ell} \in \{0,1\}$ indicating whether fast-weight layer ℓ executed, the update is
$$G_i = \big\{\widehat{w}_{i,\ell}\nabla_{\Omega_\ell}\mathcal{L}_i^{\mathrm{infer}}(\Omega_{i-1})\big\}_{\ell\in\mathcal{I}_{\mathrm{state}}}, \qquad \Omega_i = \Omega_{i-1} - \eta_i\,\mathrm{Clip}(G_i). \tag{17}$$
Clip bounds the global gradient norm of the masked collection by 1.0. During both router training and inference, the inner optimizer is stateless SGD with $\eta_i = 1.0$ and no learning-rate warm-up; Table 2 specifies the separate outer optimizer. Current predictions use Ω_{i-1}, and Ω_i becomes available to later chunks.

At an exit, $\widetilde{\alpha}_i = 1$ in value. For fixed incoming student state and hard decisions, replacing the training-time teacher comparator therefore preserves the current chunk's logits and cross-entropy value. It changes the gate derivative and can change the inner update and later fast-state trajectory. Terminal chunks run at full depth without updating fast weights; answer processing follows the full-depth protocol in §4.1.

B TRAINING CONFIGURATIONS

Backbones. Routing preserves each backbone's update granularity, inner-loss form, optimizer, and read-before-write schedule; Appendix A describes the routed TTT-E2E gradient. Our LaCT

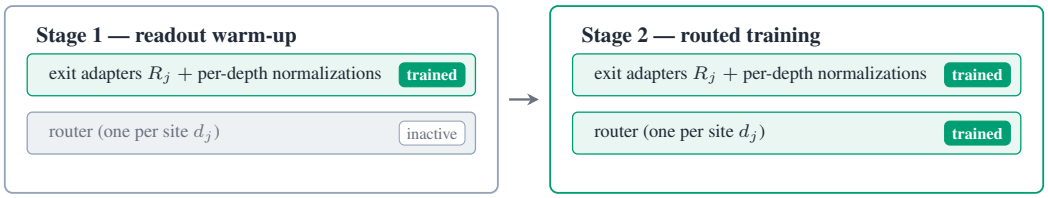


Figure 8: **Two-stage training for AdaWrite.** The backbone, final normalization, and LM head are frozen throughout. Stage 1 trains only the exit readouts, with all layers executed and all native updates enabled. Stage 2 enables hard routing and trains the router and readouts jointly.

backbone follows LaCT's architecture and fast-weight optimization design (Zhang et al., 2026): 24 layers, hidden size 1,536, 24 attention heads, four SwiGLU fast-weight heads, rank-32 learnable initial fast weights, per-token learning rates, momentum, Muon, and a 2,048-token chunk/window. We independently pretrain this 1.093B checkpoint, referred to as LaCT 1B throughout, from random initialization on ProLong-64K (Gao et al., 2025) at 65,536-token context, using the Llama-3 tokenizer (128,256 tokens): 61,440 optimizer steps with eight sequences per step (≈32.2B tokens), a 1,024-step warm-up, peak learning rate 10^{-3}, and cosine decay to 10^{-4}. This checkpoint's own frozen full-depth execution is the LaCT baseline. The TTT-E2E backbone is the released 1B checkpoint (Tandon et al., 2025): 24 layers, hidden size 2,048, sliding-window attention of width 8,192, TTT mini-batch (chunk) 1,024, and fast weights in the MLPs of the last six layers. It is pre-trained on DCLM-Baseline (Li et al., 2025) and long-context tuned on Books3, a component of the Pile (Gao et al., 2020), both at 8,192-token context, by its authors; we use it unchanged.

Router training. Table 2 lists the router and readout training settings. We use the same long-context data source as each frozen backbone: ProLong-64K for LaCT and Books3 for TTT-E2E, without task-specific or evaluation-specific training data. Because each TTT-E2E training step takes longer, we use 1,024 steps, compared with 2,048 for LaCT, while preserving Stage 1 as the first 25% of training. For both, the warm-up equals Stage 1, so the learning rate peaks when hard routing activates. The GTCF and two-stage ablations change one training factor at a time. Fixed-depth controls train one rank-128 readout at a single prefill depth under the same budget, restoring full depth for the terminal chunk and answer processing; update-schedule controls (no update and random 50%) train no parameters. The LaCT patch schedule processes approximately 4.29B token positions ($2{,}048 \times 32 \times 65{,}536$), including both stages; this counts training exposure rather than unique tokens. The efficiency proxy uses $(c_{\mathrm{pre}}, c_{\mathrm{grp}}) = (1, 1)$ for LaCT and $(18, 4)$ for TTT-E2E. Only their ratio matters. These values respectively represent a four-layer stem followed by equal-cost four-layer routed groups, and an 18-layer ordinary prefix followed by two-layer TTT groups. In the binary policy, CONTINUE always executes the complete group computation, including fast-weight application and update; these are cost components of one action, not additional router actions.

Table 2: AdaWrite hyperparameters; both backbones remain frozen. †Because the TTT-E2E pipeline is slower, it uses half as many optimization steps while retaining the same Stage-1 fraction (25%).

	LaCT 1B	**TTT-E2E 1B**
Router		
Sites K (exit depths d_j)	5 at $\{4, 8, 12, 16, 20\}$	3 at $\{18, 20, 22\}$ (TTT suffix only)
Actions per site	EXIT / CONTINUE	
Features	mean and elementwise max pooling (concatenated)	
Feature normalization	separate RMSNorms for mean/max features at each site	
Head initialization	weights $\mathcal{N}(0, 0.02^2)$; bias $(-1, +1)$, toward CONTINUE	
Temperature	1.0 (fixed)	
Decision rule	deterministic argmax; straight-through gradients in training and TTT-E2E inner updates (TTT-E2E inference: Appendix A.1)	
Exit readouts		
Adapter rank r	128 (SiLU bottleneck, zero-initialized up-projection)	
Per-exit normalization	RMSNorm; scale copied from the pretrained final norm	
Skipped-layer K/V	computed from the detached exit boundary	
Trainable parameters	2.0M ($\approx 0.18\%$)	1.6M ($\approx 0.16\%$)
Objective (equation 9)		
λ_{exit} / λ_{eff} / λ_z	0.5 / 0.5 / 10^{-3}	
GTCF	on (Stage 2 only)	
Data and sequence settings		
Frozen modules	backbone, final normalization, LM head	
Patch-training source	ProLong-64K	Books3 (the Pile)
Sequence length	65,536	8,192
Chunk (TTT mini-batch)	2,048	1,024
Global batch (sequences)	32	
Outer schedule and optimizer		
Total steps†	2,048	1,024
Stage-1 (readout-only) steps	512	256
Stage-1 fraction of total	25%	
Learning rate	10^{-3} peak $\rightarrow 10^{-4}$, cosine; linear warm-up = Stage-1 length	
Optimizer	AdamW, $\beta = (0.9, 0.95)$, weight decay 0.1; gradient clip 1.0	
Seed	42	

C Inference-FLOPs Accounting

Table 3 reports GFLOPs per processed token for each evaluation setting in Figure 3, using the convention in §4.3. For LaCT S-NIAH-1 at 128K, the depth fraction $8.2/24{=}0.34$ understates the analytical routed-to-full cost ratio of 0.45, because the accounting includes the LM head and the component costs of attention, fast-weight reads, and updates. Each updating TTT-E2E chunk takes one gradient step on its language-model loss. The backward pass traverses the executed TTT suffix and frozen head, but computes parameter gradients only for fast-weight matrices. We count the corresponding backward matrix multiplications in addition to the forward computation (windowed attention, all linear layers including fast-weight reads, the LM head, router sites, exit readouts, and substitute K/V computation in the routed model).

Accounting conventions. Costs are pooled over processed tokens, including the terminal chunk at full depth. PG-19 includes 65,536 context tokens and 256 teacher-forced continuation tokens per book; autoregressive generation is excluded. We charge the LM head at every processed token in both rows under a common convention, although some evaluations materialize fewer logits. Counts include router and exit-readout work retained by the implementation after a logical exit. Where aggregate routing records lack position information, attention costs use bounded estimates; the bound is at most 0.15% of a TTT-E2E cell total. LaCT's substitute-K/V accounting on natural text is an upper bound with an effect of at most 0.75% of the cell total. These are algorithmic FLOP estimates: kernel padding, rematerialization, replay, and small elementwise costs are excluded. The training-only teacher and ghost computations are outside this inference accounting.

Table 3: Analytical prefill FLOPs (GFLOPs per processed token) on the samples of Table 1. Counts include forward computation and native fast-weight updates (including LaCT's Muon optimizer); TTT-E2E routes only its six-layer TTT suffix. Bold marks the lower cost within each backbone.

	S-NIAH-1			LongBench v2	PG-19
	32K	64K	128K		
LaCT 1B (full)	2.53	2.54	2.55	2.55	2.55
+ AdaWrite	**1.28**	**1.18**	**1.15**	**2.43**	**2.40**
TTT-E2E 1B (full)	6.16	6.31	6.37	6.33	6.32
+ AdaWrite	**4.49**	**4.59**	**4.65**	**6.21**	**5.60**

D Detailed Experimental Results

Table 5, Table 6, and Table 7 report per-length accuracy and average executed depth for the LaCT models of §4 and §5 on RULER S-NIAH-1/2/3; Table 8 reports the same per-length breakdown for the TTT-E2E 1B backbone. The corresponding LaCT ablation trends are visualized in Figure 6 in the main paper. Depth averages are the mean of the six per-length depths, matching the convention of Table 1 and Table 4.

S-NIAH-1 is our primary retrieval analysis; S-NIAH-2/3 assess the same policy relative to its frozen backbone on essay contexts. The original LaCT study also reports low long-context accuracy on these tasks: its 760M Muon model scores 7.8% and 5.2% at 32K (Table 7 of Zhang et al., 2026). These published results provide context for the task difficulty; our comparisons use our independently trained checkpoint. The router learns through language modeling, so its decisions need not track the distribution of evidence for a later retrieval question.

Table 4: AdaWrite ablations and controls on LaCT 1B. GTCF off disables only the gate-teacher counterfactual; joint (no two-stage) removes only the readout-only first stage under the same total budget; z-loss off sets $\lambda_z = 0$ with everything else unchanged; fixed-depth models use prefill depth 20 or 22 and train a rank-128 exit readout for the same 2,048 steps. The two TTT-update ablations keep the full 24-layer forward path and use no router: no TTT update never updates the fast weights, and random 50% lets each TTT layer update on each chunk with a fair coin (pattern fixed per sample). Reference rows repeat Table 1; S-NIAH-1 is averaged over six lengths. Per-length results appear in Table 5, Table 6, and Table 7 below. Bold marks the better of full depth and AdaWrite; the last two rows are diagnostics rather than routing baselines.

	S-NIAH-1 ↑		LongBench v2 ↑		PG-19 ppl ↓	
Model	Acc	D	Acc	D	Ppl	D
LaCT 1B (full)	0.73	24.0	0.260	24.0	**16.82**	24.0
+ AdaWrite	**0.81**	9.7	**0.272**	22.4	17.11	22.1
+ GTCF off	0.56	10.4	0.262	22.2	17.60	21.7
+ joint training	0.56	14.8	0.264	23.7	17.27	23.7
+ z-loss off	0.64	10.8	0.278	19.8	18.28	18.8
Fixed d=22	0.06	22.0	0.210	22.0	25.15	22.0
Fixed d=20	0.01	20.0	0.189	20.0	44.96	20.0
No TTT updates	0.08	24.0	0.308	24.0	20.64	24.0
Random 50% updates	0.55	24.0	0.305	24.0	16.93	24.0

Table 5: LaCT 1B: RULER S-NIAH-1 accuracy and average executed depth by context length. "D" denotes average executed depth; model abbreviations follow Table 4.

	8K		16K		32K	
Model	Acc	D	Acc	D	Acc	D
Full	1.00	24.0	1.00	24.0	1.00	24.0
AdaWrite	0.92	13.84	0.91	10.28	0.94	9.06
GTCF off	0.86	15.36	0.83	11.72	0.78	9.83
Joint	1.00	23.64	0.98	20.02	0.58	13.90
z-loss off	0.94	17.04	0.93	12.16	0.84	9.96
Fixed d=22	0.20	22.0	0.10	22.0	0.02	22.0
Fixed d=20	0.03	20.0	0.01	20.0	0.00	20.0
No updates	0.23	24.0	0.13	24.0	0.03	24.0
Random 50%	0.56	24.0	0.50	24.0	0.64	24.0

	64K		96K		128K		Avg	
Model	Acc	D	Acc	D	Acc	D	Acc	D
Full	0.89	24.0	0.50	24.0	0.00	24.0	0.73	24.0
AdaWrite	0.90	8.44	0.79	8.27	0.40	8.20	0.81	9.68
GTCF off	0.64	8.83	0.19	8.47	0.07	8.32	0.56	10.42
Joint	0.27	11.16	0.32	10.17	0.18	9.62	0.56	14.75
z-loss off	0.74	8.88	0.30	8.55	0.11	8.35	0.64	10.82
Fixed d=22	0.03	22.0	0.00	22.0	0.00	22.0	0.06	22.0
Fixed d=20	0.00	20.0	0.00	20.0	0.00	20.0	0.01	20.0
No updates	0.05	24.0	0.00	24.0	0.01	24.0	0.08	24.0
Random 50%	0.65	24.0	0.56	24.0	0.38	24.0	0.55	24.0

Table 6: LaCT 1B: RULER S-NIAH-2 accuracy and average executed depth by context length. "D" denotes average executed depth; model abbreviations follow Table 4.

	8K		16K		32K	
Model	Acc	D	Acc	D	Acc	D
Full	0.62	24.0	0.19	24.0	0.04	24.0
AdaWrite	0.62	22.60	0.24	23.33	0.04	23.41
GTCF off	0.64	22.71	0.19	24.00	0.04	22.36
Joint	0.60	24.00	0.19	24.00	0.04	23.99
z-loss off	0.46	17.38	0.17	18.34	0.04	17.66
Fixed d=22	0.14	22.0	0.07	22.0	0.02	22.0
Fixed d=20	0.00	20.0	0.00	20.0	0.00	20.0
No updates	0.21	24.0	0.15	24.0	0.04	24.0
Random 50%	0.24	24.0	0.17	24.0	0.04	24.0

	64K		96K		128K		Avg	
Model	Acc	D	Acc	D	Acc	D	Acc	D
Full	0.02	24.0	0.01	24.0	0.01	24.0	0.148	24.0
AdaWrite	0.02	21.76	0.01	21.99	0.01	21.83	0.157	22.49
GTCF off	0.02	21.39	0.01	22.10	0.01	22.25	0.152	22.47
Joint	0.02	23.40	0.01	23.63	0.01	23.37	0.145	23.73
z-loss off	0.02	17.16	0.01	17.43	0.01	17.21	0.118	17.53
Fixed d=22	0.01	22.0	0.01	22.0	0.00	22.0	0.042	22.0
Fixed d=20	0.00	20.0	0.00	20.0	0.00	20.0	0.000	20.0
No updates	0.02	24.0	0.01	24.0	0.01	24.0	0.073	24.0
Random 50%	0.02	24.0	0.01	24.0	0.01	24.0	0.082	24.0

Table 7: LaCT 1B: RULER S-NIAH-3 accuracy and average executed depth by context length. "D" denotes average executed depth; model abbreviations follow Table 4.

	8K		16K		32K	
Model	Acc	D	Acc	D	Acc	D
Full	0.59	24.0	0.19	24.0	0.07	24.0
AdaWrite	0.53	23.60	0.26	23.28	0.08	23.39
GTCF off	0.58	24.00	0.19	24.00	0.08	22.28
Joint	0.58	24.00	0.19	24.00	0.07	24.00
z-loss off	0.41	17.24	0.08	18.57	0.07	18.06
Fixed d=22	0.16	22.0	0.03	22.0	0.01	22.0
Fixed d=20	0.01	20.0	0.00	20.0	0.00	20.0
No updates	0.28	24.0	0.03	24.0	0.07	24.0
Random 50%	0.38	24.0	0.11	24.0	0.08	24.0

	64K		96K		128K		Avg	
Model	Acc	D	Acc	D	Acc	D	Acc	D
Full	0.02	24.0	0.02	24.0	0.03	24.0	0.153	24.0
AdaWrite	0.02	21.78	0.02	22.55	0.03	22.23	0.157	22.80
GTCF off	0.02	21.06	0.02	22.14	0.03	22.32	0.153	22.63
Joint	0.02	23.38	0.02	23.62	0.03	23.38	0.152	23.73
z-loss off	0.02	17.30	0.02	17.61	0.03	17.36	0.105	17.69
Fixed d=22	0.01	22.0	0.01	22.0	0.02	22.0	0.040	22.0
Fixed d=20	0.00	20.0	0.00	20.0	0.00	20.0	0.002	20.0
No updates	0.02	24.0	0.03	24.0	0.00	24.0	0.072	24.0
Random 50%	0.02	24.0	0.02	24.0	0.03	24.0	0.107	24.0

Table 8: TTT-E2E 1B: RULER S-NIAH-1/2/3 accuracy and average executed depth by context length. Routing is restricted to the six-layer TTT suffix, giving an execution depth of 18–24 layers. The backbone's training context is 8K, so longer contexts probe extrapolation. "D" denotes average executed depth.

	8K		16K		32K	
Model	Acc	D	Acc	D	Acc	D
S-NIAH-1						
Full	1.00	24.0	0.47	24.0	0.17	24.0
AdaWrite	1.00	20.46	0.48	20.26	0.16	20.04
S-NIAH-2						
Full	0.99	24.0	0.21	24.0	0.09	24.0
AdaWrite	0.99	23.40	0.20	23.73	0.09	23.87
S-NIAH-3						
Full	0.77	24.0	0.15	24.0	0.08	24.0
AdaWrite	0.75	23.37	0.15	23.72	0.08	23.87

	64K		96K		128K		Avg	
Model	Acc	D	Acc	D	Acc	D	Acc	D
S-NIAH-1								
Full	0.14	24.0	0.08	24.0	0.06	24.0	0.32	24.0
AdaWrite	0.14	20.04	0.08	20.00	0.06	20.08	0.32	20.15
S-NIAH-2								
Full	0.04	24.0	0.03	24.0	0.02	24.0	0.230	24.0
AdaWrite	0.04	23.31	0.03	22.81	0.02	22.99	0.228	23.35
S-NIAH-3								
Full	0.08	24.0	0.07	24.0	0.10	24.0	0.208	24.0
AdaWrite	0.08	23.31	0.07	22.80	0.09	22.98	0.203	23.34

E Needle-Bearing Chunk Depth Analysis

Protocol. For each of the same 100 S-NIAH-1 prompts per length, we map the needle sentence to token positions and label every routed 2,048-token chunk whose span overlaps it. The terminal prefill chunk is ineligible for routing and excluded from both populations. Consequently, needles in that chunk (2/5/1 samples at 32K/64K/128K) contribute no routed needle-bearing chunk, while needles spanning a chunk boundary (1/0/1) label both chunks. The instrumented rerun exactly reproduces archived routing and accuracy at 64K and 128K. At 32K, one of 1,500 routed chunks changes from depth 8 to 24 across hardware, shifting mean depth by 0.01 and accuracy from 0.94 to 0.95 without affecting the conclusions.

Table 9 gives the chunk counts and mean depths at each length used in Figure 4.

Table 9: Needle-bearing vs. other routed-chunk depth (AdaWrite on LaCT 1B, S-NIAH-1; 100 samples per length). "At 24": share of needle-bearing chunks reaching full depth; "at 8": share of other chunks exiting at the second site.

	Needle-bearing chunks			Other routed chunks		
Length	n	Depth	At 24	n	Depth	At 8
32K	99	23.03	94%	1,401	8.09	99.4%
64K	95	22.32	89%	3,005	8.01	99.9%
128K	100	20.20	76%	6,200	8.01	99.9%

Across all three lengths, 254 of 268 full-depth routed chunks are needle-bearing. All 14 remaining full-depth chunks immediately follow a needle-bearing chunk, so deep routing remains localized to the needle neighborhood. This analysis establishes where depth is allocated; it does not by itself measure interference between fast-weight updates.

F Extended Ablation Discussion

This appendix expands the ablation summary of §5 using the aggregate and per-length results in Appendix D.

GTCF. Replacing the detached teacher with a continuation from the student's routed state (§3.4) changes average depth only modestly (10.4 vs. 9.7 layers), but lowers mean retrieval accuracy from 0.81 to 0.56 (0.79 to 0.19 at 96K, 0.40 to 0.07 at 128K; Figure 6). PG-19 perplexity increases from 17.11 to 17.60. The retrieval gap is therefore not explained by a large difference in average executed depth. This supports using an independent full-depth comparator to train the gates.

Two-stage training. Training the readouts and router jointly from the first step, with the same total training budget, reduces average S-NIAH-1 accuracy from 0.81 to 0.56, with the largest losses at 32K–128K (0.94/0.90/0.79/0.40 $\rightarrow$ 0.58/0.27/0.32/0.18), while LongBench v2 accuracy and PG-19 perplexity change little (0.264, 17.27). The joint policy is both less accurate and less efficient (14.8 vs. 9.7 layers), showing that readout alignment is important in this training setup.

Router z-loss. Setting $\lambda_z = 0$ and changing nothing else lowers S-NIAH-1 from 0.81 to 0.64 at a similar average depth (10.8 vs. 9.7 layers). The accuracy drop occurs mainly beyond the training length (96K: 0.79 to 0.30; 128K: 0.40 to 0.11), while accuracy through 16K is unchanged or slightly higher. The router also uses fewer layers on the other tasks: 18.8 layers on PG-19 (22.1 with the z-loss), 19.8 on LongBench v2 (22.4), and 17.5 to 17.7 on S-NIAH-2/3 (22.5 to 22.8), where accuracy drops to 0.118/0.105 from 0.157/0.157. PG-19 perplexity rises from 17.11 to 18.28, while LongBench v2 changes from 0.272 to 0.278. We introduced the z-loss after observing gradient-norm spikes during router training. The ablation also shows its practical value for long-context retrieval and PG-19 continuation prediction.

Fixed-depth exit. Fixing prefill depth at 20 or 22 with the same rank-128 readout and budget, while retaining full-depth terminal and answer processing, sharply reduces performance: S-NIAH-1 falls to 0.01/0.06, LongBench v2 falls below the 0.25 random-choice level, and PG-19 perplexity rises to 44.96/25.15, despite greater depth than the routed model's 9.7-layer average on S-NIAH-1. Under these matched adaptation budgets, uniform prefill truncation performs substantially worse than learned allocation.

TTT-update ablations. The last two rows of Table 4 are router-free diagnostics. Removing updates reduces S-NIAH-1 from 0.73 to 0.08 and raises PG-19 perplexity from 16.82 to 20.64, showing the value of persistent updates on these tasks. Randomly retaining half the updates improves retrieval beyond 64K over full updating (0.56 vs. 0.50 at 96K; 0.38 vs. 0.00 at 128K), but reduces shorter-context accuracy (0.64 vs. 1.00 at 32K). Learned routing reaches similar accuracy to random thinning at 128K (0.40 vs. 0.38), higher accuracy at 96K (0.79 vs. 0.56), and a higher six-length average (0.81 vs. 0.55), while executing fewer layers. Random thinning gives 16.93 PG-19 perplexity, and both update controls score 0.305–0.308 on LongBench v2 versus 0.260 for full updating and 0.272 for routing. These controls diagnose update schedules: random masks retain full forward computation and differ from prefix-depth routes. They do not isolate content-dependent placement at a matched depth distribution. The reported routing gains concern the combined execution and state-update policy, including its substitute K/V construction.

G Routing Scope on TTT-E2E: Stateful Suffix versus Whole Network

The main-text TTT-E2E model routes its stateful suffix at depths $\{18, 20, 22\}$; exits skip two, four, or six TTT layers. We compare this with whole-network routing using LaCT's five sites at depths

$\{4, 8, 12, 16, 20\}$. Its first four sites precede the TTT suffix, so exits at depth 16 or earlier skip the entire suffix and remaining ordinary Transformer layers. Both variants use the same released 1B checkpoint, Books3 patch training, 1,024 steps with the same Stage 1 fraction, GTCF, $\lambda_{\text{eff}} = 0.5$, $\lambda_z = 10^{-3}$, rank-128 readouts, and the evaluation protocol of §4.1. Table 10 reports quality and average execution depth; Table 11 reports, for each site, the fraction of chunks reaching it that exit there.

Table 10: Routing scope for AdaWrite on TTT-E2E 1B. "D": average executed depth over eligible prefill chunks (24 layers total). The TTT-suffix model is used in Table 1; whole-network routing uses sites $\{4, 8, 12, 16, 20\}$ with otherwise identical training. Bold compares routed models; the full-depth baseline is shared. S-NIAH-1 averages six lengths.

	S-NIAH-1 ↑		LongBench v2 ↑		PG-19 ppl ↓	
Model	Acc	D	Acc	D	Ppl	D
TTT-E2E 1B (full)	0.32	24.0	0.249	24.0	11.84	24.0
+ AdaWrite (TTT suffix)	**0.32**	20.1	0.237	23.7	**12.24**	22.4
+ AdaWrite (whole network)	0.30	24.0	**0.245**	20.5	12.69	19.6

Table 11: Per-site exit rates for AdaWrite on TTT-E2E 1B, pooled over eligible chunks and conditioned on reaching each site. Chunks that exit are excluded from later sites. Both layouts list site depths.

Scope	Task	Exit rate at site (%)				
TTT suffix only	site depth	18	20	22		
	S-NIAH-1	0.0	98.5	0.0		
	LongBench v2	1.7	3.5	1.9		
	PG-19	2.6	37.9	0.3		
Whole network	site depth	4	8	12	16	20
	S-NIAH-1	0.1	0.0	0.0	0.0	0.0
	LongBench v2	8.2	12.1	0.2	0.2	1.3
	PG-19	12.1	11.7	0.1	1.5	7.8

Where the two routers exit. The two layouts allocate depth differently. Suffix routing reduces depth mainly on S-NIAH-1: 98.5% of chunks reaching depth 20 exit there, so most chunks update the first two of the six stateful layers. On natural text it stays deeper, averaging 23.7 layers on LongBench v2 and 22.4 on PG-19. Whole-network routing almost never exits on S-NIAH-1 (24.0 layers after rounding), but stops after four or eight layers on roughly one fifth of natural-text chunks. The rates in Table 11 are conditional on reaching each site; combining the first two gives approximately 19% of LongBench v2 chunks and 22% of PG-19 chunks. These shallow exits omit both ordinary Transformer layers and the entire TTT suffix.

Quality and depth. Whole-network routing gives lower retrieval accuracy: S-NIAH-1 falls to 0.92 from 1.00 at 8K and averages 0.30 instead of 0.32; S-NIAH-2/3 fall to 0.203/0.165 from 0.228/0.203. PG-19 perplexity is 12.69, compared with 12.24 for suffix routing and 11.84 for the full-depth baseline. Whole-network routing gives slightly higher LongBench v2 accuracy (0.245 vs. 0.237). Allowing shallower exits therefore reduces depth on natural text, with different effects on task performance. Suffix routing retains the ordinary Transformer prefix and selects write depth in increments of two stateful layers.

Why route the stateful suffix. Suffix routing saves deep writes on S-NIAH-1, whereas whole-network routing remains near full depth. This supports routing around fast-weight layers (§3.2), where each decision controls memory transitions. Adaptive computation in the ordinary Transformer prefix remains a separate direction.