UNIVERSAL TEST-TIME TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent Test-Time Training (TTT) architectures compress context into fast weights that are updated online and queried as memory. Existing TTT designs keep this memory private to each layer: it recurs only over time, and depth merely indexes L separate memories. We argue that memory ownership need not be tied to depth, and introduce *Universal Test-Time Training (uTTT)*, in which all layers read and write one shared memory while retaining layer-specific backbone parameters. The shared memory thus recurs over two dimensions, time and depth, with chunks and layers as their units: a write by a deep layer in one chunk can be read by a shallow layer in the next. We instantiate this idea as *uTTT-MoE* and *uTTT-Dense*. uTTT-MoE routes each token head to a few experts in a pool shared by all layers; uTTT-Dense applies the whole shared memory at every layer without routing. In language modeling, uTTT-MoE reaches 15.5 and 27.9 RULER accuracy at 124M and 760M, 2.6 and 2.1 points above its layer-private counterpart at equal state and active compute, the highest among tested bounded-state models, with per-token loss matching or beating full attention. In novel view synthesis, sharing at fixed per-layer compute gains 0.92 dB in view-23 object PSNR in routed models and 0.76 dB in dense models.

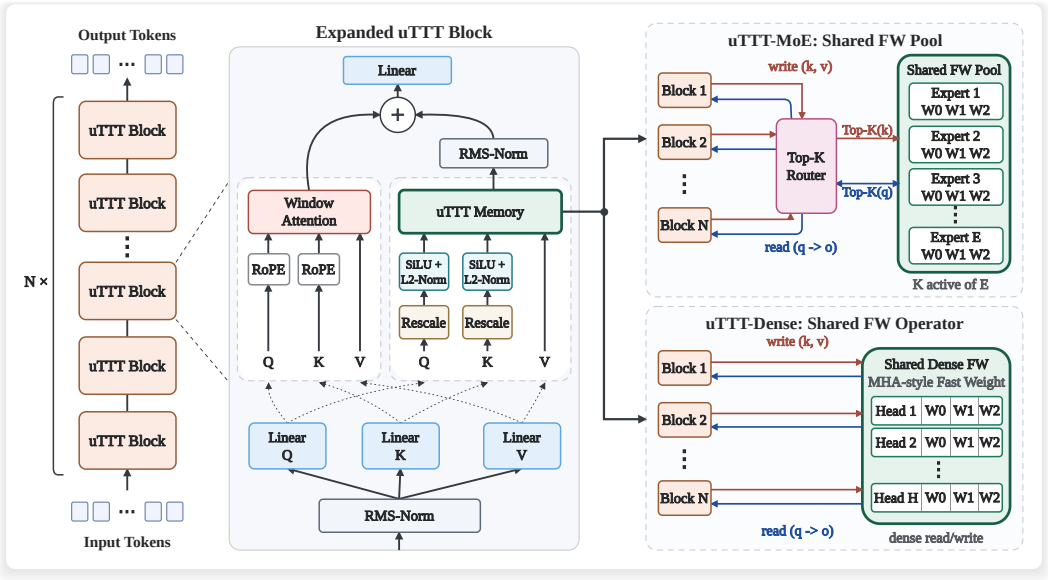


Figure 1: **Universal Test-Time Training.** A block (*middle*) keeps its normalization, projections, and window attention private and sends q, k, v to a memory with state shared across the stack (*left*). The state (*right*) is a pool of E fast-weight experts under a top-K router (uTTT-MoE, top) or dense fast weights applied in full (uTTT-Dense, bottom). Blue arrows read (2); red arrows write (3).

1 INTRODUCTION

Long-context models trade explicit memory for compressed state. Attention keeps the full key-value history at growing cost (Vaswani et al., 2017), while recurrent networks, linear attention, and state-space models keep a bounded state (Hochreiter & Schmidhuber, 1997; Katharopoulos

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.

et al., 2020; Gu & Dao, 2024; Dao & Gu, 2024; Yang et al., 2024). Recent Test-Time Training (TTT) architectures store context in *fast weights* that are optimized online during the forward pass and queried as memory (Hinton & Plaut, 1987; Schmidhuber, 1992; Schlag et al., 2021; Sun et al., 2025; Behrouz et al., 2025); large-chunk TTT amortizes these updates to make nonlinear fast-weight memory practical at long context (Zhang et al., 2026).

Once fast weights serve as memory, a design question is *who owns them*. Layerwise KV-binding TTT (Sun et al., 2025) and LaCT (Zhang et al., 2026) give each layer its own state, written using its key-value pairs and read by its queries. TTT-E2E (Tandon et al., 2025) updates selected layers' MLP weights under one next-token prediction loss, but still keeps separate weight tensors across depth. In all of these designs, layer ℓ's memory takes only layer ℓ's writes, so it evolves along time alone and depth only multiplies memories: a layer can use only its own share of the budget, and no layer can read state written by another.

In this work, we call fast weights *universal* when all layers read from and write to one shared state or one shared pool of states. *Universal Test-Time Training (uTTT)* builds on such fast weights while keeping backbone parameters layer-specific (Figure 1). Its memory then evolves along both time and depth, measured in chunks along time and in layers along depth, and layer-private TTT is the special case that partitions it by layer. This opens two routes that layer-private TTT lacks: a shallow layer can directly read what a deep layer wrote in the previous chunk, a *top-down route*, and the outer task loss at that read backpropagates through the write, so the slow weights learn how layers at different depths build and use one memory (Sections 2 and 3.4). Unlike prior sharing across depth, uTTT shares a bounded fast-weight memory that every layer writes by test-time optimization within each sequence, and keeps these writes in the training graph.

uTTT comes in two forms. *uTTT-MoE* gives every layer a shared pool of E expert states, from which each token head selects K per read or write. Its control, *TTT-MoE*, uses the same router but restricts each layer to its own E/L experts, so sharing widens each layer's choices from E/L to E experts at equal total state and active experts. *uTTT-Dense* applies one dense shared state at every layer. Compared with *TTT-Dense*, the layer-private dense design of LaCT, it lets every layer apply the whole state at a fixed budget, raising fast-weight compute, or matches per-layer compute with only $1/L$ of the state. Because all layers write the same tensors, each chunk's updates also merge into one operation, which speeds up training.

Related work. TTT uses fast weights as a context-dependent memory state (Sun et al., 2023; 2025; von Oswald et al., 2026; Wang et al., 2026). MoM routes among linear memory states (Du et al., 2026); our TTT-MoE control adds routing to LaCT's nonlinear fast weights. Shared memory also appears in Feedback Transformers (Fan et al., 2020), block-recurrent models (Ju et al., 2022; Hutchins et al., 2022), and cross-layer key–value caches (Sun et al., 2024; Wu & Tu, 2024; Brandon et al., 2024). Universal Transformers, ALBERT and their MoE variants share slow parameters across depth (Dehghani et al., 2019; Lan et al., 2020; Tan et al., 2023; Csordás et al., 2024). UniPool and MoUE likewise share trained expert parameters (Huang et al., 2026; Chen et al., 2026); MoUE additionally shares an online-updated routing state. Appendix A gives the extended comparison.

Contributions.

- We propose uTTT, a TTT architecture whose fast-weight memory is shared by all layers, in routed (uTTT-MoE) and dense (uTTT-Dense) forms, with one causal update per chunk for all layers' writes and a layer-private routed control, TTT-MoE, that isolates sharing from routing.
- We find that sharing outperforms layer-private TTT at equal or smaller state. uTTT-MoE gains 2.6 and 2.1 RULER points over TTT-MoE at 124M and 760M, leading the tested bounded-state models (Figure 3), and sharing adds 0.92 and 0.76 dB object PSNR in routed and dense view synthesis. uTTT-MoE also trains up to $1.39\times$ faster than TTT-MoE.
- We show through analysis and interventions that sharing unifies time and depth in one memory recurrence, with layer-private TTT as its partitioned special case (Section 3.4). A read can depend on every layer's earlier writes: probes detect these cross-layer routes in trained models, and removing them, by knocking out cross-layer reads at inference or cutting cross-depth gradients in training, degrades performance (Figure 4).

2 PRELIMINARY

LaCT and fast-weight state. Large-chunk test-time training (LaCT) (Zhang et al., 2026) equips each layer with a module that stores context in fast weights. We use *memory* for this function and *state* for its concrete, updatable representation. We partition each sequence into contiguous token blocks, called chunks, $C_1, \ldots, C_N$, and process them in causal order. At layer ℓ and fast-weight head h (a *layer-head*), slow-parameter projections produce query, key and value vectors $\mathbf{q}_{\ell,h,i}, \mathbf{k}_{\ell,h,i}, \mathbf{v}_{\ell,h,i} \in \mathbb{R}^{d_h}$ for token i.

Suppressing these indices, one SwiGLU (Shazeer, 2020) module has state $W = (W_0, W_1, W_2)$, with $W_0, W_2 \in \mathbb{R}^{d_{\text{fw}} \times d_h}$ and $W_1 \in \mathbb{R}^{d_h \times d_{\text{fw}}}$, and read function $f_W(\mathbf{x}) = W_1[\text{SiLU}(W_0\mathbf{x}) \odot (W_2\mathbf{x})]$. Here d_{fw} is its internal width and $\odot$ elementwise multiplication. A model's fast-weight state collects these matrix triples; its recurrent implementation also carries master accumulators and, where used, momentum buffers. LaCT keeps each layer's state private. Chunk-level writes amortize update costs; a read itself leaves the state unchanged and supplies features used for task predictions.

Inner loop: associative writes. A read evaluates f_W on a query. A write differentiates the associative loss $\ell_{\text{in}}(W; \mathbf{k}, \mathbf{v}) = -\langle f_W(\mathbf{k}), \mathbf{v} \rangle$. Within chunk c, reads use the state produced after chunk $c-1$; a boundary update aggregates token contributions:

$$\begin{aligned} \mathbf{o}_i &= f_{W^{(c-1)}}(\mathbf{q}_i), \qquad i \in C_c, \\ W^{(c)} &= \text{Update}\left(W^{(c-1)},\ \nabla_W \sum_{i \in C_c} \eta_i \ell_{\text{in}}(W; \mathbf{k}_i, \mathbf{v}_i) \Big|_{W = W^{(c-1)}} \right). \end{aligned} \tag{1}$$

The effective step sizes η_i come from slow-parameter predictors; matrix-specific coefficients and fixed reductions are suppressed. Update denotes a stateful optimizer carrying master weights and any momentum buffers, with predicted momentum coefficients also implicit. Language models use momentum, Muon and row renormalization; NVS omits momentum and Muon (Appendix C.2). Inner updates run during training and inference. Fast weights reset to their learned initialization between sequences; the final language-modeling chunk needs no write.

Outer loop: learning to write and read. During training, the task loss $\mathcal{L}_{\text{task}}$ supervises predictions produced using the read outputs. Along the state chain, backpropagation follows $\mathcal{L}_{\text{task}} \to \mathbf{o} \to W^{(c-1)} \to \text{Update}_{s<c} \to \theta$, reaching slow parameters θ through earlier differentiable writes, including their inner-loss gradient computation. This trains the slow projections, coefficient predictors and initialization to produce useful future reads. The task loss supplies the outer learning signal; the associative loss determines each online write. AdamW applies these outer gradients across training sequences. At inference the slow parameters remain fixed: online inner updates still adapt the state, but no outer task-loss update is performed. Thus the two loops optimize different quantities: the inner loop changes sequence-specific state; the outer loop learns how that adaptation supports the task.

3 UNIVERSAL FAST WEIGHTS

Universal Test-Time Training (uTTT) shares the fast-weight state across depth while retaining layer-specific backbone weights, projections and local attention (Figure 1).

3.1 SHARED FAST WEIGHTS

The layer-private construction in Section 2 gives each layer its own state. uTTT instead partitions the L layers into p contiguous groups and gives each group one shared state, a collection of the SwiGLU matrix triples defined above. Every layer in a group reads that state and contributes write inputs to it. At $p = L$ the groups are layer-private; at $p = 1$ all depths share one state. The following instantiations specify how the modules f_W access its components.

We use an *aggregated* write schedule: all layers process chunk c using the fast weights present before that chunk, and their inner-loss gradients are evaluated at those same weights. At the boundary, contributions are combined and one optimizer update is applied per group, including its recurrent

buffers. No layer reads a write from its current chunk. Only write-enabled chunks update state; NVS target chunks containing only camera poses are read-only. Sharing changes which layers contribute to and use the state while retaining this causal protocol.

State size counts fast-weight elements, excluding optimizer buffers and attention caches. Total state is stored per sequence; reachable state is available to one layer, and active state is read or written by one token at one layer-head (ℓ, h). We distinguish these quantities because sharing can change their relationship without changing the read network.

3.2 uTTT-MoE

The routed instantiation stores a pool $\mathcal{B} = \{W_e\}_{e=1}^{E}$ of expert states, each a SwiGLU matrix triple with the dimensions of Section 2. A group owns E/p experts. Each layer-head uses separate read and write projections to select top-K experts from that group's pool, denoted $\mathcal{R}^r_{\ell,h}(\mathbf{q})$ and $\mathcal{R}^w_{\ell,h}(\mathbf{k})$. The scores compare normalized features with learned expert vectors. Gates g^r, g^w are selected, scaled softmax probabilities from these *slow-parameter routers*; they are not parameters of the online-updated expert state.

$$\mathbf{o}_{\ell,h,i} = \sum_{e \in \mathcal{R}^r_{\ell,h}(\mathbf{q}_{\ell,h,i})} g^r_{\ell,h,i,e}\, f_{W_e^{(c-1)}}(\mathbf{q}_{\ell,h,i}). \tag{2}$$

Let $\mathcal{I}_{c,e}$ collect the layer-head-token triples whose write selects expert e in chunk c. Its aggregate gradient and update are

$$\begin{aligned} G_{c,e} &= \nabla_{W_e} \sum_{(\ell,h,i)\in\mathcal{I}_{c,e}} g^w_{\ell,h,i,e}\eta_{\ell,h,i}\ell_{\text{in}}(W_e; \mathbf{k}_{\ell,h,i}, \mathbf{v}_{\ell,h,i})\Big|_{W_e=W_e^{(c-1)}}, \\ W_e^{(c)} &= \text{Update}\big(W_e^{(c-1)}, G_{c,e}\big). \end{aligned} \tag{3}$$

The inner derivative holds keys, values, gates and predicted step sizes fixed; outer backpropagation retains their producer graph. As in (1), η absorbs fixed reductions and Update carries optimizer buffers (Appendix C.2). Read and write selections can differ. At $p = L$ this is **TTT-MoE**; at $p = 1$ it is **uTTT-MoE**. With expert size u, total state stays Eu and active state per token head stays Ku, while each layer's candidate pool grows from E/L to E experts. TTT-MoE controls for routing; load balancing acts on utilization across the shared pool (Appendix E.1).

3.3 uTTT-Dense

A dense group holds one state with H head blocks, each applied in full without expert selection. Each block reads its query using f_W and receives writes from the corresponding heads of the group's layers. At $p = L$ this is **TTT-Dense**, with L layer-private states of size S (S/H active per token head); at $p = 1$ it is **uTTT-Dense**.

Two controls separate sharing from widening. Sharing a state of size S preserves per-token fast-weight compute while reducing total state from LS to S. Widening the shared state to LS instead preserves total state and increases active state per head from S/H to LS/H, multiplying fast-weight compute by L. For intermediate p, these paths store pS or LS respectively. Main language comparisons match total state ($Eu = LS$); view synthesis tests both controls. Table C1 gives the full matching conditions.

3.4 Unifying Time and Depth

Sharing fast-weight state extends the two learning loops in Section 2 across depth. In the *inner loop*, layers jointly update their group's state at write-enabled chunk boundaries. A deep layer's write in chunk c can then support a shallower layer's read in chunk $c + 1$, a *top-down route* absent from private state (Figure 2a). In the *outer loop*, task-loss gradients from later reads propagate through these inner updates to train earlier writes at other depths. Figure 2b shows the restricted state-chain Jacobian used by this backward pass; full training also retains query and hidden-state producer paths.

Writing S_c for the complete recurrent state (fast weights, master accumulators, optimizer buffers and attention caches), the aggregated model obeys $S_c = f_\theta(S_{c-1}, X_c)$. Each transition runs all layers on

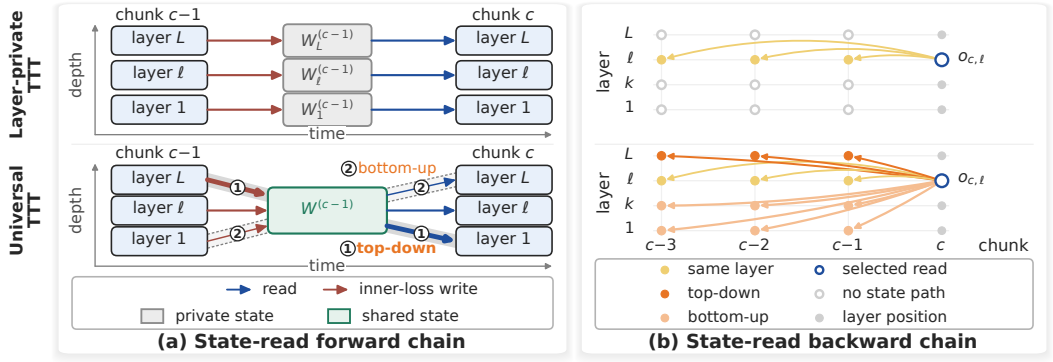


Figure 2: **State-read forward and backward chains.** Layer-private TTT (top) and universal TTT (bottom). *(a)* Forward reads use the chunk-start state. Inner-loop writes prepare the state for later reads during both training and inference (Section 2). Bands trace top-down (1) and bottom-up (2) routes. *(b)* The restricted read-output Jacobian traces backward through historical inner updates to write inputs; it is a factor of outer task-loss backpropagation. Each update includes the inner-loss gradient computation. Queries and predicted coefficient producers are held fixed here; full training retains those additional paths. Hollow nodes lack this restricted state-chain path, not necessarily all full-model gradient. Colors identify the forward writer-to-reader relation; arrows point backward. Routing determines which paths are active (Appendix I.1).

chunk X_c using the chunk-start fast weights and combines each group's writes at the boundary. Our runtime therefore advances chunk by chunk; this execution order differs from the two optimization loops above (Appendix C.3). A direct routed path requires writer and reader to select a common expert. Proposition 1 formalizes the gradient reach, and the forward-preserving gradient cut tests its training role (Appendix I.2).

4 Experiments: Language Modeling

Long-context language modeling tests how well a fixed fast-weight state retains information beyond the local attention window.

4.1 Setup

Data, chunks, and metrics. Following the large-chunk TTT language-modeling protocol (Zhang et al., 2026), we train autoregressive models at 32K context on Long-Data-Collections (Together AI, 2023) at 124M and 760M parameters. The token stream is split into causal chunks of 4,096 tokens, with sliding-window attention over 4,096 tokens for local context, so a token reads only fast weights written by earlier chunks whichever schedule writes them (Section 3.4). We report two quantities. Per-token loss (PTL) is the mean next-token loss over positions at or beyond 30K in 32K-token sequences of Books3 from The Pile (Gao et al., 2020), following the same protocol and the per-position convention of the Forgetting Transformer (Lin et al., 2025); RULER retrieval accuracy (Hsieh et al., 2024) is averaged over the 4K, 8K, 16K, and 32K settings. The two are not redundant: a model can lower average loss without improving retrieval. Appendix E.1 reports six training seeds of uTTT-MoE at 124M.

Models. The four fast-weight models in the main comparison have the *same total fast-weight state* (Table C1). TTT-Dense implements the layer-private dense design of LaCT, also reported from its published code. TTT-MoE routes each token head into a layer-local pool of four experts ($E/L = 4$, $K = 1$); uTTT-MoE shares the pool across depth, with $E = 48$ at 124M and $E = 96$ at 760M, keeping $K = 1$ and a router per layer. Both routed models use loss-free balancing and the aggregated schedule; uTTT-MoE aggregates utilization statistics across the pool and backpropagates the outer task loss through state reads and differentiable inner-loss writes. All four share the tokenizer, chunk size, sliding-window attention, depth, width, optimizer and token budget. The seven baselines are

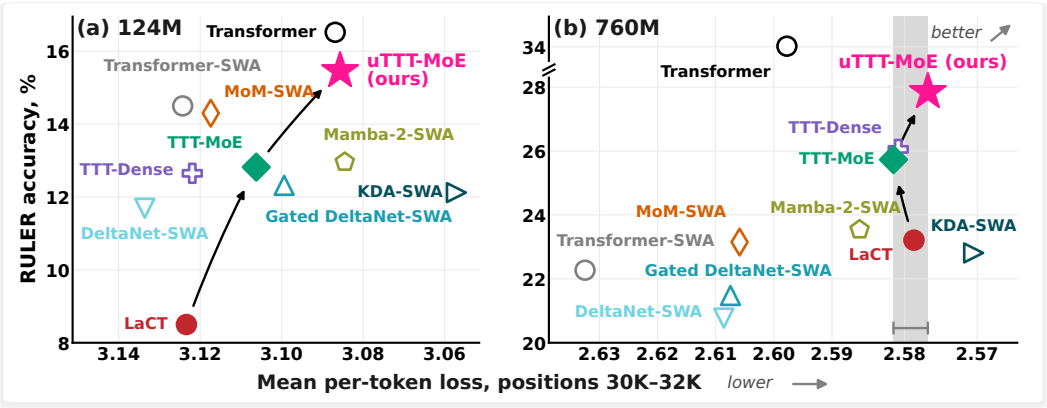


Figure 3: **Loss–retrieval Pareto plot**, at (a) 124M and (b) 760M parameters. One point per model (Section 4); the horizontal axis is mean per-token loss beyond 30K in 32K-token sequences, and the vertical axis is mean RULER accuracy over 4K–32K. Arrows run through the fast-weight models from LaCT to uTTT-MoE. The shaded band in (b) spans the four fast-weight models, whose losses differ by 0.0047 nat. The vertical-axis break in (b) marks a compressed empty stretch.

Table 1: **Long-context language modeling at 32K.** PTL is the mean next-token loss at positions $\geq$ 30K on Books3, and the five columns beside it are RULER accuracy (%): the mean over 4K, 8K, 16K and 32K, then each of those lengths. Bold marks the best value in each column among the bounded-state models; the full-attention Transformer (†) keeps a key–value cache that grows with context and is excluded from that comparison. The lower block holds the fast-weight models, of which uTTT-MoE is the model this paper proposes.

	124M						760M					
Model	PTL ↓	RULER ↑	4K	8K	16K	32K	PTL ↓	RULER ↑	4K	8K	16K	32K
Baselines												
Transformer †	3.0869	16.52	23.82	20.24	13.10	8.93	2.5978	34.16	41.70	37.95	33.36	23.62
Transformer-SWA	3.1243	14.50	**26.93**	14.73	9.94	6.41	2.6325	22.27	45.23	22.64	12.67	8.57
Gated DeltaNet-SWA	3.0994	12.32	23.78	12.93	8.30	4.26	2.6075	21.48	40.40	23.12	13.66	8.76
DeltaNet-SWA	3.1336	11.69	22.52	11.23	7.63	5.36	2.6086	20.77	43.43	20.56	11.79	7.28
KDA-SWA	**3.0571**	12.12	20.86	12.98	8.36	6.28	**2.5705**	22.81	43.47	24.17	14.65	8.96
Mamba-2-SWA	3.0845	12.96	24.21	13.93	8.32	5.37	2.5862	23.54	48.42	25.49	12.60	7.65
MoM-SWA	3.1174	14.30	26.11	14.10	10.41	6.58	2.6059	23.15	43.35	23.69	15.97	9.60
Fast-weight family												
LaCT (published)	3.1234	8.50	13.75	11.37	5.93	2.94	2.5787	23.21	43.65	23.97	14.29	**10.92**
TTT-Dense	3.1219	12.65	26.31	13.58	6.00	4.72	2.5809	26.05	47.73	30.07	16.49	9.92
TTT-MoE	3.1062	12.82	25.35	13.26	7.60	5.08	2.5815	25.73	45.27	29.44	**17.85**	10.35
uTTT-MoE (ours)	3.0857	**15.46**	26.50	**17.32**	**11.23**	**6.78**	2.5768	**27.85**	**54.15**	**31.00**	16.03	10.23

full-attention Transformer, Transformer-SWA, DeltaNet-SWA (Yang et al., 2024), Gated DeltaNet-SWA (Yang et al., 2025), KDA-SWA (Kimi Team, 2025), Mamba-2-SWA (Dao & Gu, 2024) and MoM-SWA (Du et al., 2026); all SWA models use the same 4,096-token window. Appendix D gives the configurations, and Appendix E.2 evaluates the dense shared branch at 124M, where it retrieves as well as TTT-Dense with one sixth to one half of the state.

4.2 Results

uTTT-MoE leads bounded-state RULER at both scales while matching or beating full attention's per-token loss (Table 1, Figure 3). At 124M, KDA-SWA and Mamba-2-SWA attain lower loss but retrieve 3.3 and 2.5 points less; at 760M only KDA-SWA does, retrieving 5.0 points less. Appendix E gives retrieval-task and per-position loss curves.

Sharing adds 2.6 and 2.1 RULER points over TTT-MoE at 124M and 760M, with the same state and no loss penalty; TTT-MoE stays near TTT-Dense. uTTT-MoE leads TTT-MoE at every length at

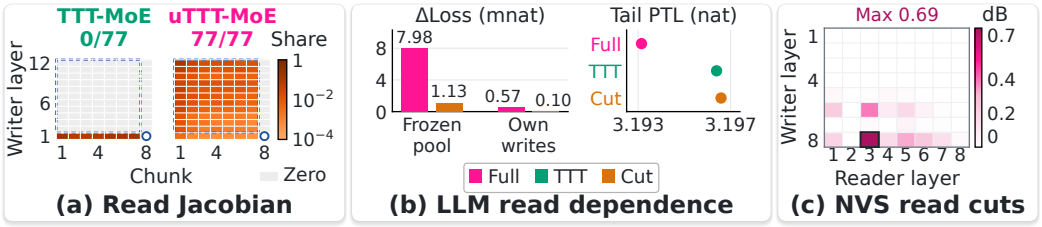


Figure 4: **State-read backward and forward dependence.** LLM diagnostics use 124M models and individual Books3 documents' first 32K tokens. *(a)* Squared-gradient shares from a layer-1, chunk-8 read-output Jacobian probe, normalized per document and averaged over 300 documents. This probes a local factor of outer task-loss backpropagation. Colors are logarithmic; gray is zero. Boxed counts require nonzero gradients in every document; circles mark the read. *(b)* On 1,024 documents, bars show loss increases after freezing state writes or reading only own writes, relative to each model's normal predictions over chunks 2–8; dots show unmodified tail PTL at positions $\geq$ 30,000, measured on these unpacked documents and therefore not on the scale of Table 1. Full uses full uTTT-MoE backpropagation; Cut stops cross-layer read-to-write gradients during training; TTT is TTT-MoE. *(c)* PSNR drop after removing one writer–reader route in uTTT-MoE-e64-p1, averaged over 1,019 GSO objects with 23 target views each. Diagonal routes are included. Colors are linear, with drops below 0.01 dB white; the outline marks the maximum. Panels (b,c) use fixed trained checkpoints.

124M; at 760M it leads at 4K and 8K and trails by less than two points at 16K and 32K (Table 1). Its loss advantage over Transformer-SWA persists to 128K, but retrieval past 32K falls to the window's level for every bounded-state model (Appendix F).

Load balancing keeps the pool in use: without it the busiest expert receives 5.9$\times$ its even share of the token heads at 124M and 9.6$\times$ at 760M, against 1.4$\times$ and 1.8$\times$ for the reported uTTT-MoE (Appendix E.1). At 124M removing it costs uTTT-MoE 4.6 RULER points and TTT-MoE 2.8, both to below TTT-Dense. At 760M the unbalanced run has the higher mean, 28.46 against 27.85, but the lead comes from one easy needle task (S-NIAH-1), it loses 8 to 9 points on five others, and it reads mostly from 22 of the 96 experts. We balance the pool rather than each layer: the pool is what is shared, and requiring every layer to use every expert equally would remove the depth allocation that sharing is for (Appendix E.1). We use the loss-free bias for both routed models; the auxiliary-loss results are in Appendix E.1.

The read-output probe's local Jacobian reaches all 77 earlier deep-layer write locations in uTTT-MoE and none in TTT-MoE across 300 Books3 documents (Figure 4a). This probes a local factor used by outer task-loss backpropagation. On 1,024 documents, freezing state writes raises loss by 7.98 mnat in Full and 1.13 mnat in Cut, indicating weaker read dependence on written state in Cut (Figure 4b). Appendix I gives the probe, training and intervention protocols.

4.3 Measured cost

uTTT-MoE trains fastest in every setting of Figure 5a and reaches TTT-Dense's final training loss in 1,540 H100 GPU-hours, 15% fewer than TTT-Dense and 27% fewer than TTT-MoE (Figure 5c). Both routed models use the same fused kernels (Table G1). uTTT-MoE aggregates cross-layer contributions into one update of the shared expert pool per non-final chunk boundary, while TTT-MoE updates its layer-private expert pools separately. This permits fewer, larger kernel launches; the observed speedup and step-time breakdown (Figure 5b) are consistent with reduced launch overhead. Appendix G reports kernel ablations, sweeps over the number of experts, inference and memory measurements.

5 Experiments: Novel View Synthesis

Novel view synthesis tests a bounded scene memory: attention stays within each view, so information from other views must pass through the fast-weight state. Training uses eleven writes and

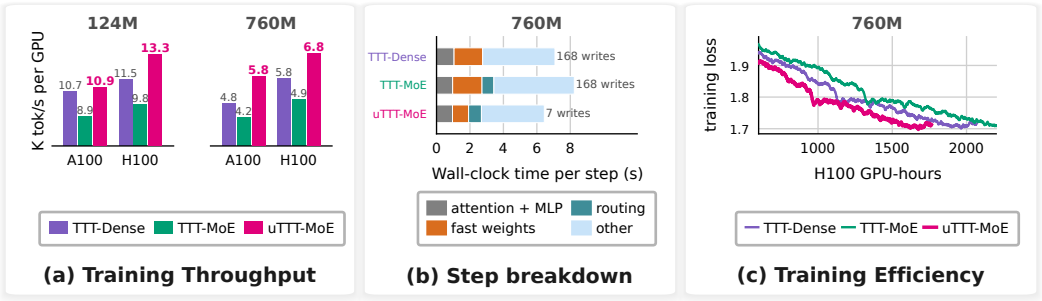


Figure 5: **Measured training efficiency.** *(a)* Training throughput of the three ownership regimes on one node of eight A100 or H100 GPUs (32K context, batch 1 per GPU), in thousands of tokens per second per GPU. *(b)* Where a 760M training step's time goes (one A100, batch 1); attention and MLP are identical across the three models, and *other* is the backward pass and the optimizer. *(c)* Training loss of the three 760M models against the H100 GPU-hours spent.

evaluation uses 23, testing beyond the trained horizon. We separate sharing at fixed per-layer compute from compute scaling at fixed total state.

5.1 Setup

Task and protocol. Object models train on Objaverse (Deitke et al., 2023) and evaluate on Google Scanned Objects (Downs et al., 2022) at 256×256; scene models use DL3DV (Ling et al., 2024), with about 10K training and 140 test scenes. Following LVSM (Jin et al., 2025), a posed image patchified with ray encodings forms a source chunk that reads and writes the state; a target camera encoded by rays alone forms a read-only chunk that renders the held-out view. Reads and writes use (2)–(3). Training views and their order are sampled randomly.

Models. The L=8 layers form p pools (p=8 private, p=1 universal). Each routed pool holds $e = E/p$ experts, with a active per token head (the K of Section 3.2); dense pools have width w. An active expert and a width unit cost the same per-layer compute, but a width unit stores eight times as much state: 64 experts match width 8. Names read uTTT-MoE-ee-pp-aa (a=1 omitted) and uTTT-Dense-ww-pp; e8-p8 and w1-p8 are TTT-MoE-e8 and TTT-Dense. Routed models use aggregated writes. Full attention accumulates keys and values across views with other settings matched. Table D5 gives the recipe shared by objects and scenes except for the dataset.

5.2 From private pools to one shared pool

Row A of Figure 6 merges eight private pools into four, two and one at fixed per-layer compute. Routed models retain 64 experts at fixed parameters, state and active compute; dense models retain width 1 per pool, reducing total state eightfold. Full sharing performs best in both families, gaining 0.92 dB for routed models and 0.76 dB for dense models at view 23, with disjoint intervals for the all-view means (Table H1). The curves separate as views accumulate and the gains persist beyond the trained horizon; the dense gain needs neither routing nor more state.

A shared pool of only 8 experts also beats the private pools by 0.99 dB at view 23, with disjoint intervals for the all-view means, despite storing one eighth of the state and offering the same eight candidates per token head. This isolates shared writes from a wider routing choice. Appendix H repeats the sharing progression at 32 experts; Appendix I.3 examines reuse across views and layers.

Single-route knockouts reveal the strongest dependence on top-down state-read routes: removing writes from layer 8 to reads at layer 3 reduces PSNR by 0.69 dB, compared with at most 0.06 dB for same-layer routes (Figure 4c). Appendix I.2 gives the intervention protocol.

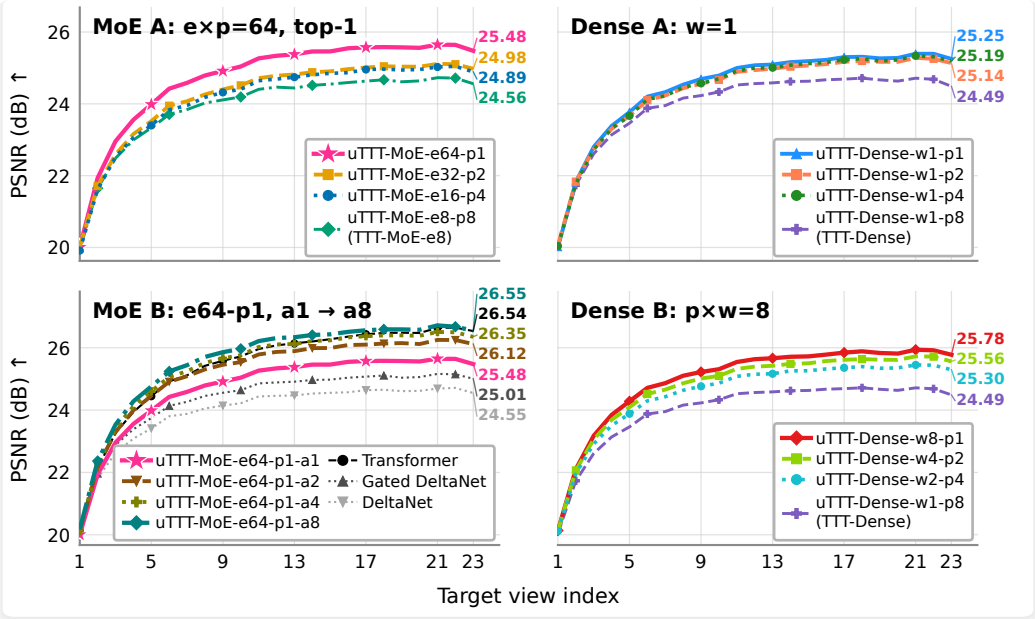


Figure 6: **Sharing (row A) and compute scaling (row B) on Objaverse/GSO**: PSNR against the target-view index, i.e. the number of posed views written into the pool; the number at the right end of each curve is its value at view 23. The e, p, w, and a in model names are defined in Section 5.1. Row A holds per-layer fast-weight compute fixed (a=1, w=1) and widens sharing from p=8 to p=1; row B holds total fast-weight state fixed (64 experts, width 8) and raises per-layer compute from one unit to eight. The MoE panel of row B also shows the three baselines at the same protocol: the full-attention reference (black dashed) and the linear-attention Gated DeltaNet and DeltaNet (grey dotted). Full metrics and 95% bootstrap intervals are in Table H1.

5.3 More compute at fixed state

Row B fixes total state at 64 experts or width 8 and matches per-layer compute: routed models activate a=1, 2, 4, 8 experts; dense models widen to w=1, 2, 4, 8 as pools merge (p=8, 4, 2, 1), changing sharing as well. Routed view-23 PSNR improves monotonically by 1.07 dB. The dense gain is 1.29 dB: 0.76 from sharing and 0.53 from widening, consistent with the width sweep at full sharing (Appendix H). Routed models lead their compute-matched dense partners at every level, by 0.77 dB at eight units, with disjoint all-view intervals. At a=8, the fixed state matches full attention at view 23 (26.55 versus 26.54 dB) and leads at earlier views (Table H1). All three metrics improve; LPIPS falls from 0.185 for TTT-Dense to 0.158 for uTTT-MoE at a=8.

Appendix H confirms monotone activation scaling at 32 experts (25.30 → 26.39). Growing the pool from 8 to 64 experts at a=1 changes view-23 PSNR by at most 0.25 dB without a consistent trend. On DL3DV, increasing a=1 to a=8 raises all-view PSNR from 16.12 to 16.66.

6 Conclusion

Fast-weight memory need not be owned by individual layers. uTTT shares it across depth, so the memory evolves along time and depth together, and layer-private TTT becomes one partition of a larger design space. At equal or smaller state, uTTT improves both language modeling and view synthesis, its routed form trains faster, and trained models depend on the cross-layer routes that sharing opens. Depth then adds to test-time optimization instead of multiplying separate memories.

AI USE STATEMENT

We used large language models to assist with revising and polishing the manuscript. The authors take responsibility for the final content of this paper, including its scientific claims and reported results.

REPRODUCIBILITY STATEMENT

Sections 2–3 define the model and its fast-weight read/write operations. Appendix C specifies the inner updates and write schedules, Appendix D records model configurations and training hyperparameters, and Appendix B identifies dataset sources and access terms. Evaluation protocols are described in Sections 4 and 5, with additional language-modeling, length-extrapolation, efficiency and novel-view-synthesis results in Appendices E–H. Appendix I provides the gradient-reach proof and the probe, training-control and state-read intervention protocols.

REFERENCES

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In *Advances in Neural Information Processing Systems*, 2016.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time. In *Advances in Neural Information Processing Systems*, 2025.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Vincent-Pierre Berges, Barlas Oğuz, Daniel Haziza, Wen-tau Yih, Luke Zettlemoyer, and Gargi Ghosh. Memory layers at scale. In *International Conference on Machine Learning*, 2025.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. Reducing transformer key-value cache size with cross-layer attention. In *Advances in Neural Information Processing Systems*, 2024.

Yilong Chen, Naibin Gu, Junyuan Shang, Zhenyu Zhang, Yuchen Feng, Jiawei Sheng, Tingwen Liu, Shuohuan Wang, Yu Sun, Hua Wu, and Haifeng Wang. Mixture of universal experts: Scaling virtual width via depth-width transformation. *arXiv preprint arXiv:2603.04971*, 2026.

Kevin Clark, Kelvin Guu, Ming-Wei Chang, Panupong Pasupat, Geoffrey Hinton, and Mohammad Norouzi. Meta-learning fast weight language models. In *Conference on Empirical Methods in Natural Language Processing*, 2022.

Róbert Csordás, Kazuki Irie, Jürgen Schmidhuber, Christopher Potts, and Christopher D. Manning. MoEUT: Mixture-of-experts universal transformers. In *Advances in Neural Information Processing Systems*, 2024.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning*, 2024.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019.

Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3D objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Laura Downs, Anthony Francis, Nate Koenig, Brandon Kinman, Ryan Hickman, Krista Reymann, Thomas B McHugh, and Vincent Vanhoucke. Google Scanned Objects: A high-quality dataset of 3D scanned household items. In *IEEE International Conference on Robotics and Automation*, 2022.

Jusen Du, Weigao Sun, Disen Lan, Jiaxi Hu, Tao Zhang, and Yu Cheng. MoM: Linear sequence modeling with mixture-of-memories. In *International Conference on Learning Representations*, 2026.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing some limitations of transformers with feedback memory. *arXiv preprint arXiv:2002.09402*, 2020.

William Fedus, Barret Zoph, and Noam Shazeer. Switch Transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017.

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. *arXiv preprint arXiv:2101.00027*, 2020.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Language Modeling*, 2024.

Zijin Gu, Tatiana Likhomanenko, Vimal Thilak, Jason Ramapuram, and Navdeep Jaitly. Path-constrained mixture-of-experts. *arXiv preprint arXiv:2603.18297*, 2026.

Geoffrey E. Hinton and David C. Plaut. Using fast weights to deblur old memories. In *Annual Conference of the Cognitive Science Society*, 1987.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Yicong Hong, Kai Zhang, Jiuxiang Gu, Sai Bi, Yang Zhou, Difan Liu, Feng Liu, Kalyan Sunkavalli, Trung Bui, and Hao Tan. LRM: Large reconstruction model for single image to 3D. In *International Conference on Learning Representations*, 2024.

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

Minbin Huang, Han Shi, Chuanyang Zheng, Yimeng Wu, Guoxuan Chen, Xintong Yu, Yichun Yin, and Hong Cheng. UniPool: A globally shared expert pool for mixture-of-experts. *arXiv preprint arXiv:2605.06665*, 2026.

DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. Block-recurrent transformers. In *Advances in Neural Information Processing Systems*, 2022.

Martin Jaggi. Tying the loop – tied expert layers in mixture-of-experts language models. *arXiv preprint arXiv:2606.16825*, 2026.

Haian Jin, Hanwen Jiang, Hao Tan, Kai Zhang, Sai Bi, Tianyuan Zhang, Fujun Luan, Noah Snavely, and Zexiang Xu. LVSM: A large view synthesis model with minimal 3D inductive bias. In *International Conference on Learning Representations*, 2025.

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

Da Ju, Stephen Roller, Sainbayar Sukhbaatar, and Jason Weston. Staircase attention for recurrent processing of sequences. In *Advances in Neural Information Processing Systems*, 2022.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, 2020.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3D Gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4):1–14, 2023.

Kimi Team. Kimi Linear: An expressive, efficient attention architecture. *arXiv preprint arXiv:2510.26692*, 2025.

Ben Krause, Emmanuel Kahembwe, Iain Murray, and Steve Renals. Dynamic evaluation of neural sequence models. In *International Conference on Machine Learning*, 2018.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Large memory layers with product keys. In *Advances in Neural Information Processing Systems*, 2019.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020.

Zhixuan Lin, Evgenii Nikishin, Xu He, and Aaron Courville. Forgetting transformer: Softmax attention with a forget gate. In *International Conference on Learning Representations*, 2025.

Lu Ling, Yichen Sheng, Zhi Tu, Wentian Zhao, Cheng Xin, Kun Wan, Lantao Yu, Qianyu Guo, Zixun Yu, Yawen Lu, Xuanmao Li, Xingpeng Sun, Rohan Ashok, Aniruddha Mukherjee, Hao Kang, Xiangrui Kong, Gang Hua, Tianyi Zhang, Bedrich Benes, and Aniket Bera. DL3DV-10K: A large-scale scene dataset for deep learning-based 3D vision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, 2010.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*, 2020.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for SQuAD. In *Annual Meeting of the Association for Computational Linguistics*, 2018.

Mehdi S. M. Sajjadi, Henning Meyer, Etienne Pot, Urs Bergmann, Klaus Greff, Noha Radwan, Suhani Vora, Mario Lučić, Daniel Duckworth, Alexey Dosovitskiy, Jakob Uszkoreit, Thomas Funkhouser, and Andrea Tagliasacchi. Scene representation transformer: Geometry-free novel view synthesis through set-latent scene representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

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

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

Noam Shazeer. GLU variants improve Transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

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. In *International Conference on Machine Learning*, 2020.

Yu Sun, Xinhao Li, Karan Dalal, Chloe Hsu, Sanmi Koyejo, Carlos Guestrin, Xiaolong Wang, Tatsunori Hashimoto, and Xinlei Chen. Learning to (Learn at test time). *arXiv preprint arXiv:2310.13807*, 2023.

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. In *International Conference on Machine Learning*, 2025.

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

Lintang Sutawika, Hailey Schoelkopf, Leo Gao, Baber Abbasi, Stella Biderman, Jonathan Tow, ben fattori, Charles Lovering, farzanehnakhaee70, Jason Phang, Anish Thite, Fazz, Thomas Wang, Niklas, Aflah, sdtblck, nopperl, gakada, tttyuntian, researcher2, James A. Michaelov, Julen Etxaniz, Taksh Kothari, Chris, Hanwool Albert Lee, Leonid Sinev, Janna, Zdeněk Kasner, Kiersten Stokes, and Khalid. EleutherAI/lm-evaluation-harness: v0.4.13. Zenodo, version v0.4.13, 2026. URL https://zenodo.org/records/22214970.

Shawn Tan, Yikang Shen, Zhenfang Chen, Aaron Courville, and Chuang Gan. Sparse universal transformer. In *Conference on Empirical Methods in Natural Language Processing*, 2023.

Zheyue Tan, Zhiyuan Li, Tao Yuan, Dong Zhou, Weilin Liu, Yueqing Zhuang, Yadong Li, Guowei Niu, Cheng Qin, Zhuyu Yao, Congyi Liu, Haiyang Xu, Boxun Li, Guohao Dai, Bo Zhao, and Yu Wang. ReXMoE: Reusing experts with minimal overhead in mixture-of-experts. *arXiv preprint arXiv:2510.17483*, 2025.

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

Together AI. Long-Data-Collections. Hugging Face dataset, no longer publicly accessible; data mixture documented in the LLaMA-2-7B-32K model card, 2023. URL https://huggingface.co/togethercomputer/LLaMA-2-7B-32K#training-and-fine-tuning.

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

Johannes von Oswald, Nino Scherrer, Seijin Kobayashi, Luca Versari, Songlin Yang, Maximilian Schlegel, Kaitlin Maile, Yanick Schimpf, Oliver Sieberling, Alexander Meulemans, Guillaume Lajoie, Rif A. Saurous, Charlotte Frenkel, Razvan Pascanu, Blaise Agüera y Arcas, and João Sacramento. MesaNet: Sequence modeling by locally optimal test-time training. In *International Conference on Learning Representations*, 2026.

Ke Alexander Wang, Jiaxin Shi, and Emily B. Fox. Test-time regression: a unifying framework for designing sequence models with associative memory. *Journal of Machine Learning Research*, 27 (169):1–41, 2026.

Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. *arXiv preprint arXiv:2408.15664*, 2024.

Qianqian Wang, Yifei Zhang, Aleksander Holynski, Alexei A. Efros, and Angjoo Kanazawa. Continuous 3D perception model with persistent state. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Haoyi Wu and Kewei Tu. Layer-condensed KV cache for efficient inference of large language models. In *Annual Meeting of the Association for Computational Linguistics*, 2024.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems*, 2024.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations*, 2025.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing*, 2018.

Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. pixelNeRF: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, 2019.

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 *International Conference on Learning Representations*, 2026.

Appendix Contents

A Extended Related Work **16**

A.1 Fast weights and test-time training 16

A.2 Sharing parameters and experts across depth 16

A.3 Novel view synthesis 16

B Datasets and Licenses **16**

C Fast-Weight Preliminaries and the Design Space **17**

C.1 Pool ownership, routing, and matching 17

C.2 Read network and write update 18

C.3 Write schedules and execution order 19

D Model Configurations **19**

D.1 Language modeling 19

D.2 Novel view synthesis 21

E Language-Modeling Experiments **22**

E.1 Load Balancing 22

E.2 A Dense Shared State at 124M 23

F Length Extrapolation **26**

G Efficiency measurements **27**

H Additional Novel View Synthesis Comparisons **28**

H.1 Sharing and Compute on Scenes 28

H.2 Total Experts and Pool Size 28

H.3 Every Configuration 29

H.4 Write Schedules 29

I Analysis of Shared Memory **33**

I.1 Mechanism: recurrence and cross-depth reach 33

I.2 Validation: probe, training control and interventions 34

I.3 Behavior: expert reuse by redundant tokens 37

A Extended Related Work

A.1 Fast weights and test-time training

Fast weights update rapidly on recent data while slow weights change rarely (Hinton & Plaut, 1987; Schmidhuber, 1992; Ba et al., 2016); Fast Weight Layers express dynamic-evaluation updates (Mikolov et al., 2010; Krause et al., 2018) as linear attention trained end to end (Clark et al., 2022). Besides the long-memory use of Section 1, test-time training can improve generalization on novel instances by optimizing a self-supervised objective at test time (Sun et al., 2020). Our TTT layers are meta-learned: the chunk writes form the inner loop, and ordinary outer-loop training learns the initial fast weights, as in MAML (Finn et al., 2017), together with the step-size and, where used, momentum-coefficient predictors.

A.2 Sharing parameters and experts across depth

Sparse mixture-of-experts layers route each token to one or a few experts (Shazeer et al., 2017; Fedus et al., 2022). Universal Transformers (Dehghani et al., 2019) and ALBERT (Lan et al., 2020) tie a block's parameters across layers; SUT and MoEUT tie the whole layer, experts and router included, per depth step (Tan et al., 2023) or per repeated group of layers (Csordás et al., 2024). ReXMoE shares experts across adjacent layers (Tan et al., 2025); other designs tie the expert sub-block within groups of layers (Jaggi, 2026) or share router parameters within blocks of consecutive layers while keeping per-layer experts (Gu et al., 2026). UniPool replaces layer-private expert sets with one shared pool, addressed by per-layer routers and balanced at the pool level (Huang et al., 2026). MoUE keeps layer-local experts beside a pool of universal experts, of which each group of consecutive layers reaches a window that shifts with depth, counting the expert paths this opens across depth as virtual width; its router also scores experts against a state matrix, which MoUE calls fast weights over depth, shared by layers with the same reachable experts and updated after routing, without backpropagation, from the current batch's keys and detached routing targets (Chen et al., 2026).

Product-key memory and Memory Layers use sparse lookup into key–value parameters learned during ordinary training (Lample et al., 2019; Berges et al., 2025), and Memory Layers shares these parameters across its memory layers. Apart from MoUE's routing state, everything the designs above route to, look up or share stays fixed at inference; uTTT instead shares the fast-weight memory itself, which all layers write within each sequence and whose updates are retained in the training graph (Section 3.4).

A.3 Novel view synthesis

Scene representations are fitted per scene (Mildenhall et al., 2020; Kerbl et al., 2023), which is itself test-time optimization, or inferred from one or a few images by a prior learned across scenes (Yu et al., 2021; Hong et al., 2024). SRT and encoder–decoder LVSM replace the 3D representation with a set of latent tokens that a decoder queries with target rays (Sajjadi et al., 2022; Jin et al., 2025); decoder-only LVSM, whose tokenization Section 5 adopts, drops even that, running input-view and target-ray tokens through one transformer. LaCT applies large-chunk, layer-private fast weights to novel view synthesis (Zhang et al., 2026), and the novel-view block of Section 5 follows its layer design (Appendix D). A geometry transformer can also fold unposed images one by one into a persistent state (Wang et al., 2025).

B Datasets and Licenses

Table B1 lists the external datasets used in the experiments with their license or access terms.

Table B1: External datasets, with license or access terms.

Asset	Use in this paper	License or access
Objaverse (Deitke et al., 2023)	NVS training	ODC-By v1.0; per-object CC licenses, some non-commercial
Google Scanned Objects (Downs et al., 2022)	NVS evaluation	CC BY 4.0
DL3DV-10K (Ling et al., 2024)	NVS training and evaluation	CC BY-NC 4.0; gated access
Long-Data-Collections (Together AI, 2023)	LM training	Mixed sources, each under its own license
Books3 from The Pile (Gao et al., 2020)	LM evaluation (Section 4; Appendices E, F and I)	Copyrighted books; withdrawn from its original host in 2023
RULER (Hsieh et al., 2024)	Retrieval evaluation, via the LM Evaluation Harness (Sutawika et al., 2026)	Apache-2.0 (harness: MIT); some needle tasks use Paul Graham essays as filler, the QA tasks use the two datasets below, the rest are synthetic
SQuAD 2.0 (Rajpurkar et al., 2018)	RULER single-hop QA (development set)	CC BY-SA 4.0
HotpotQA (Yang et al., 2018)	RULER multi-hop QA (development set, distractor setting)	CC BY-SA 4.0

C Fast-Weight Preliminaries and the Design Space

C.1 Pool ownership, routing, and matching

Let $\mathcal{B} = \{W_e\}_{e=1}^{E}$ be the pool of Section 3, each W_e one SwiGLU triple, either a routed expert or a dense head block. The group of layer ℓ owns a disjoint part of the pool, its *visible domain* $\Omega_\ell \subseteq \{1, \dots, E\}$, and each layer-head (ℓ, h) routes into it, $\mathcal{R}_{\ell,h}(z) \subseteq \Omega_\ell$. Groups are equal-sized, with integer p satisfying $1 \le p \le L$ and $p \mid L$; routed pools also require $p \mid E$ and integer K with $1 \le K \le E/p$. Table C1 lists both families along p.

In a dense pool the route is a fixed singleton, $\mathcal{R}_{\ell,h} \equiv \{e(\ell, h)\}$ with unit gate, where $e(\ell, h)$ is the group's block for head h. In a routed pool, each layer-head normalizes its routing feature $\hat{z}_{\ell,h}$ per head, the query for reads and the key for writes, and routes each token head to the K experts with the largest inner product against learned expert addresses $r_{\ell,e}$,

$$\mathcal{R}_{\ell,h}(z) \;=\; \mathrm{TopK}\left(\left\{\hat{z}_{\ell,h}^{\top} r_{\ell,e}\right\}_{e \in \Omega_\ell},\; K\right); \tag{4}$$

the language models learn the addresses per layer, while NVS shares them across layers, $r_{\ell,e} = r_e$. The softmax over these logits, kept for the selected experts and multiplied by the routing-weight scale of (16), gives the gates of (2) and (3).

Table C1: **Designs along the group count p.** H counts fast-weight heads and S is one private dense layer's full state. E counts SwiGLU triples of size $|W_e|$; layer ℓ reaches $|\Omega_\ell|\,|W_e|$, while Active counts the state applied by one token head. The fixed-compute dense row stores pS at unchanged per-token fast-weight compute; the other dense rows preserve total state LS, and routed rows preserve Eu. The uTTT-Dense row is size-matched. A write-enabled chunk ends in p group updates under the aggregated schedule and L layer updates under the sequential one (Appendix C.3).

Design	p	E	$\lvert W_e\rvert$	$\lvert\Omega_\ell\rvert$	Reachable	Active	Memories	Writes per chunk
TTT-Dense (LaCT)	L	LH	S/H	H	S	S/H	L	L
Dense, fixed compute	p	pH	S/H	H	S	S/H	p	p or L
Dense, fixed state	p	pH	$LS/(pH)$	H	LS/p	$LS/(pH)$	p	p or L
uTTT-Dense	1	H	LS/H	H	LS	LS/H	1	1 or L
TTT-MoE	L	E	u	E/L	Eu/L	Ku	L	L
Routed, p groups	p	E	u	E/p	Eu/p	Ku	p	p or L
uTTT-MoE	1	E	u	E	Eu	Ku	1	1 or L

C.2 READ NETWORK AND WRITE UPDATE

Expert blocks use $d_{\text{fw}} = d_h$, while dense widening increases d_{fw}. For matrix $m \in \{0, 1, 2\}$ of W_e, the aggregated write direction is

$$D_{c,e}^{(m)} = \rho \sum_{(\ell,h,i)\in\mathcal{I}_{c,e}} g_{\ell,h,i,e}^{w}\, \eta_{\ell,h,i}^{(m)} \nabla_{W_e^{(m)}} \left[v_{\ell,h,i}^{\top} f_{W_e}(k_{\ell,h,i}) \right] \Big|_{W_e = W_e^{(c-1)}}. \tag{5}$$

It has the opposite sign to the loss gradient supplied to Update in (3), which suppresses the matrix index on η and absorbs the reduction factor ρ into it. LaCT, TTT-Dense and both routed language models use $\rho = 1$, as does routed NVS. Language-modeling uTTT-Dense averages across layers, giving $\rho = 1/L$; dense NVS averages over chunk tokens, giving $\rho = 1/|C_c|$, and sums across sharing layers.

Predicted step sizes and momentum. The input $\bar{x}_{\ell,i}$ to the coefficient predictors is the normalized layer input supplied to attention in the main language-modeling configurations. In NVS, it is obtained by applying an additional RMS normalization without learned affine parameters ($\epsilon = 10^{-5}$) to the same layer's normalized memory input. Step sizes take the form

$$\eta_{\ell,h,i}^{(m)} = \mathrm{softplus}\Big(\psi_{\ell,h}^{(m)}(\bar{x}_{\ell,i}) + \mathrm{softplus}^{-1}(\eta_0) \Big), \qquad \eta_0 = 10^{-3}\ \text{(LM)}, \quad 10^{-2}\ \text{(NVS)}. \tag{6}$$

The ψ are learned affine projections, bias-free in NVS. In the routed language models, a layer predicts a scalar momentum coefficient for each token and averages over the chunk. A shared pool further averages over the layers $\mathcal{G}$ of its group:

$$\mu_{c,\ell} = \frac{1}{|C_c|} \sum_{i\in C_c} \sigma(\pi_\ell(\bar{x}_{\ell,i})), \qquad \mu_c = \frac{1}{|\mathcal{G}|} \sum_{\ell\in\mathcal{G}} \mu_{c,\ell}. \tag{7}$$

Here σ is the sigmoid and π_ℓ a learned linear projection. This mean is over the pool's layers and tokens, independent of which experts they select; $|\mathcal{G}| = 1$ in a private pool. Dense language models instead predict a separate coefficient per head and average it over tokens.

Routed language-modeling update. Let U be the master weights, W the fast weights used by both reads and write-gradient evaluations, and M the momentum buffers. Each sequence starts from the learned matrices U_{init}, parameterized as in Table D3, with $U_{c=0} = W_{c=0} = U_{\text{init}}$ and $M_{c=0} = 0$; the initial matrices are not row-normalized. For each expert and each matrix, with these indices suppressed, the boundary update is

$$\begin{aligned} M_c &= D_c + \mu_c M_{c-1}, \qquad B_c = U_{c-1} + \mathcal{O}_5(M_c), \\ U_c &= \mathcal{P}_{\text{row}}(B_c; U_{c-1}), \quad W_c = U_c. \end{aligned} \tag{8}$$

The row rescaling uses the pre-update master weights as its reference:

$$\left[\mathcal{P}_{\text{row}}(B; R)\right]_{j,:} = \frac{\|R_{j,:}\|_2}{\|B_{j,:}\|_2 + 10^{-5}} B_{j,:}. \tag{9}$$

The buffer stores M_c before the Muon transform $\mathcal{O}_5$. The released routed configurations use a boundary multiplier of one: the token step sizes have already entered D_c and are not multiplied in again. An expert with no new write can still change through its previous momentum.

Five-step Muon transform. For each update matrix, transpose M if it has more rows than columns, then form $X_0 = M/(\|M\|_F + 10^{-7})$ and iterate

$$A_t = X_t X_t^{\top}, \qquad X_{t+1} = a_t X_t + (b_t A_t + c_t A_t^2) X_t, \quad t = 0, \ldots, 4. \tag{10}$$

It uses the five coefficient triples

a_t	b_t	c_t
4.0848	−6.8946	2.9270
3.9505	−6.3029	2.6377
3.7418	−5.5913	2.3037
2.8769	−3.1427	1.2046
2.8366	−3.0525	1.2012

and transposes X_5 back when necessary. This is the matrix orthogonalization step used by the LaCT-derived write implementation with Muon (Jordan et al., 2024); it is applied separately to the three matrices of each expert or head, with the width-block convention for language-modeling uTTT-Dense described below.

Dense language modeling and NVS. TTT-Dense retains an unnormalized master accumulator: after the same momentum and Muon steps, $U_c = U_{c-1} + \mathcal{O}_5(M_c)$ and $W_c = \mathcal{P}_{\text{row}}(U_c; W_{c-1})$. Thus its rescaling is applied to the fast copy using the previous fast weights' row norms, while (8) rescales the master itself. LaCT uses the sequence-initial row norms in this rescaling, replacing W_{c-1} with the initial fast weights W_{init} as the reference. Language-modeling uTTT-Dense uses the unnormalized-master transition of TTT-Dense, with each head's write directions and momentum coefficients averaged across layers. At width multiplier r, Muon and row rescaling act independently on its r square width blocks. The NVS configurations omit both momentum and Muon and use

$$U_c = U_{c-1} + D_c, \qquad (W_c)_{j,:} = \frac{(U_c)_{j,:}}{\max(\|(U_c)_{j,:}\|_2, 10^{-5})}. \tag{11}$$

At each sequence start, $U_{c=0}$ is the learned initial master state and $W_{c=0}$ is its row-normalized copy, using the normalization in (11). The unnormalized master persists across chunks; the normalized fast copy is used for reads and for evaluating the next write gradient. Only source-view chunks write, while target pose-only chunks read.

C.3 Write schedules and execution order

Under both schedules all layers read the chunk-start state $W^{(c-1)}$; they differ in how the L per-layer contributions $G_{c,\ell}$ are applied at the chunk boundary. The *sequential* schedule applies them as L successive steps, evaluating each gradient at the state left by the preceding update,

$$W^{(c,\ell)} = \text{Update}\left(W^{(c,\ell-1)}, G_{c,\ell}\right), \qquad W^{(c,0)} = W^{(c-1)}, \quad W^{(c)} = W^{(c,L)}. \tag{12}$$

The *aggregated* schedule of Section 3.1 evaluates all gradients at $W^{(c-1)}$ and applies their sum in one step,

$$W^{(c)} = \text{Update}\left(W^{(c-1)}, \textstyle\sum_{\ell=1}^{L} G_{c,\ell}\right). \tag{13}$$

With p groups, both schedules act within each group. A group's next chunk waits for all of its layers, so our runtime runs chunk-first (Section 3.4); private states also permit LaCT's layer-first order. The aggregated schedule reduces the optimizer updates per write-enabled chunk from L to p; routing and raw-gradient computation remain layer-specific. We train with it; Appendix H.4 compares the two schedules on view synthesis.

D Model Configurations

This appendix records how every model we train is built, read from the training code and configurations. For each experiment it gives the settings all compared models share, then what distinguishes each model, and, for language modeling, the fast-weight block itself.

D.1 Language modeling

Every language model follows one recipe (Table D1) and differs only in the token mixer beside the 4K window (Table D2); Table D3 specifies the fast-weight block, whose read and update equations are given in Appendix C.2.

How the models are aligned. Table D2 groups the models by what they are matched in and cites the work each mixer comes from. Gated DeltaNet-SWA's narrower FFN pays exactly for its output-gate projection, and Mamba-2-SWA's state size $N = 0.75\,d$ is set so that its state equals the fast-weight state, while its FFN is narrowed to $1{,}152/2{,}304$ to pay for the window branch's own q, k, v projections, which every other mixer reuses from the window, so it is matched in parameters as well. MoM-SWA routes each token to one of four Gated DeltaNet memories that share one k, v projection; it keeps Gated DeltaNet-SWA's FFN and pays for its router with $36{,}864$ further weights at 124M and $147{,}456$ at 760M, so it matches the other linear-attention rows in parameters while carrying four times their state.

Table D1: Settings shared by every language model. Where one value spans both columns it holds at both scales.

	124M	760M
Width d / layers L / attention heads	768 / 12 / 12	1536 / 24 / 24
Attention head dimension	64	
FFN (SwiGLU (Shazeer, 2020)) hidden width	2,048	4,096
Normalization	pre-norm RMSNorm (Zhang & Sennrich, 2019)	
Vocabulary	32,000; untied embeddings	
RoPE (Su et al., 2024) θ	10^6	
Sliding window	4,096 (every model but the Transformer)	
Context length	32,768	
Data	Long-Data-Collections	
Optimizer	AdamW (Loshchilov & Hutter, 2019)	
β, ε, weight decay	$(0.9, 0.95)$, 10^{-15}, 0.1	
Learning rate	10^{-3}; 1,024 warmup steps, cosine decay to 10^{-4}	
Gradient clip	1.0	
Global batch	32 sequences (1,048,576 tokens)	
Training steps	10,240	40,960
Training tokens	10.7B	42.9B

Table D2: What distinguishes each language model. Every mixer but full attention runs beside the 4K window of Table D1 and, except Mamba-2's, reuses its q, k, v projections; the linear-attention mixers apply SiLU and L2 to q, k and use no short convolution. State is what a layer carries across chunks; uTTT-MoE pools it, $4L$ experts shared by all layers. Parameters exclude the input embedding and output head (24.6M / 49.2M each).

			Non-emb. params	
Model	FFN width	State / layer	124M	760M
Attention references				
Transformer (Vaswani et al., 2017)	2,048 / 4,096	KV cache	84.95M	679.6M
Transformer-SWA (Beltagy et al., 2020)	2,048 / 4,096	—	84.95M	679.6M
Linear attention, matched in parameters				
DeltaNet-SWA (Yang et al., 2024)	2,048 / 4,096	$64\,d$	85.10M	680.6M
Gated DeltaNet-SWA (Yang et al., 2025)	1,792 / 3,584	$64\,d$	85.21M	681.5M
KDA-SWA (Kimi Team, 2025)	1,984 / 4,032	$64\,d$	85.71M	683.0M
MoM-SWA (Du et al., 2026)	1,792 / 3,584	$4 \times 64\,d$	85.25M	681.6M
State space, matched in parameters				
Mamba-2-SWA (Dao & Gu, 2024)	1,152 / 2,304	$0.75\,d^2$	85.23M	681.1M
Fast-weight models, matched in state				
LaCT (Zhang et al., 2026)	2,048 / 4,096	$0.75\,d^2$	88.13M	699.3M
TTT-Dense	2,048 / 4,096	$0.75\,d^2$	88.13M	699.3M
TTT-MoE	2,048 / 4,096	$0.75\,d^2$	88.11M	699.2M
uTTT-MoE	2,048 / 4,096	$0.75\,d^2$	88.21M	700.1M

Table D3: The fast-weight block (d: model width; two values are 124M / 760M). LaCT and TTT-Dense compute the same block in a different order: LaCT runs the whole sequence through one layer at a time, TTT-Dense one chunk at a time through all layers.

	LaCT, TTT-Dense	TTT-MoE	uTTT-MoE
Heads × width	$4 \times d/4$		
Function	SwiGLU; initial W_0, W_2: rank $32 + 0.5\,I$		
q, k, v	window attention's, pre-RoPE; SiLU; L2, then RoPE on q, k		
Step size	per token, head, and matrix; base 10^{-3}		
Update	momentum, Muon (Jordan et al., 2024) (5 steps), row renormalization		
Momentum coefficient	per head	per chunk	per chunk
Chunk	4,096 tokens; read, then write		
Pool	per layer	4 experts / layer	$4L$ experts, shared
Router	—	softmax top-1, per layer	softmax top-1, per layer
Load balancing	—	loss-free 10^{-5}	loss-free, $3{\times}10^{-5}$ / 10^{-3}
Write	own pool	own pool	aggregated, once per chunk

Other language models. uTTT-Dense (Appendix E.2) replaces the per-layer pools of TTT-Dense with one dense state shared by all layers, r times as wide as one layer's ($r \in \{1, 2, 4, 6\}$ in Table E4), with a full-rank learned initial state and an aggregated chunk schedule; each layer keeps its own step-size, momentum, gate, and output modules, and the recipe is that of Table D1. Its layer averaging and blockwise update are specified in Appendix C.2. The seven 124M models of Appendix F are separate training runs: five use the configurations of Transformer, Transformer-SWA, LaCT, TTT-MoE, and uTTT-MoE, and the two 64K-trained attention references change only the sequence length and train 16 sequences per step, so each step still sees 1,048,576 tokens and the budget is unchanged.

D.2 Novel view synthesis

Every novel-view-synthesis model of Section 5 shares one backbone and recipe (Table D5); its modules change only the fast-weight pool, as defined in Section 5.1 and Appendix H. Object- and scene-level models differ only in data. Object models are trained for roughly 71B tokens and scene models for roughly 58B — a global batch of 128 sequences, each of 22 chunks of 1,024 tokens (11 posed views and 11 target cameras) — in a single stage at fixed resolution and view count, on NVIDIA A100 GPUs.

Table D4: What distinguishes each novel-view model. All share Table D5. Parameters count every trainable weight, including learnable initial states; the fast-weight row is for 64 total experts or total width 8 (36.25M outside the state, 0.79M initial state).

Model	FFN width	State / layer	Params
References			
Transformer	1,536	KV cache (+1,024 tokens per view)	27.76M
DeltaNet	1,584	8×64^2, delta rule	37.04M
Gated DeltaNet	1,408	8×64^2, gated delta rule	37.01M
Fast-weight models of Section 5			
64 total experts or total width 8	1,536	$8 \times 3 \times 64^2$ per width unit	37.04M

The two instantiations. Figure 1 draws the language-modeling block: the window attention and the fast-weight memory run in parallel on one set of q, k, v projections, and their outputs are summed before one output projection (Table D3). The novel-view block follows the LaCT recipe for view synthesis (Zhang et al., 2026) instead: three sequential pre-norm sublayers, a memory with its own q, k, v projections, eight fast-weight heads of dimension 64 rather than four of dimension $d/4$, and an update without momentum or Muon (Table D5). What the two share is the axis this paper studies, which layers may read and write a pool.

Reference models. Three references keep the backbone, block order, recipe and global batch of Table D5 and change only the memory sublayer and the FFN width that pays for it (Table D4). The Transformer, an LVSM-style model (Jin et al., 2025), drops the fast-weight sublayer: each layer attends to the current view and to the cached keys and values of every earlier posed view, so its memory grows by 1,024 tokens per view, and target chunks never enter the cache. DeltaNet and Gated DeltaNet replace the fast-weight sublayer by a linear-attention memory with the same 8×64 heads, SiLU and L2 on q, k and no short convolution, updated once per view chunk by the delta rule, $U \leftarrow e^{\bar{g}}\, U + \frac{1}{n} \sum_i \beta_i \,(v_i - W k_i)\, k_i^\top$ over the n tokens of the chunk, whose rows are L2-normalized into the read copy W as in (11); $\bar{g}$ is the chunk mean of the forget gate, Gated DeltaNet also has a gated output, and DeltaNet has neither gate; both start from a learnable state, as LaCT does. Their FFN widths set them to the parameter count of the fast-weight models once the learnable initial states are included. The Transformer is evaluated at step 24,000 (objects) and 20,000 (scenes), the linear-attention models at their final step (24,476 and 20,046), all under the protocol of Appendix H.

Table D5: Settings shared by every novel-view-synthesis fast-weight model of Section 5; 36.25M parameters outside the fast-weight state.

Width / layers / attention heads	512 / 8 / 8
Block	attention → fast-weight read → SwiGLU FFN (1,536)
Attention	within one view
Input	256^2, patch 8: 1,024 tokens per view
Views	training 12 (11 posed + 11 target chunks); evaluation 24
Fast-weight heads	8×64
Unit of state	expert: 3×64^2 weights, shared by heads; width unit: 8 experts
q, k, v	SiLU; L2 on q, k
Step size	per token, head, and matrix; base 0.01
Update	no momentum or Muon; rows L2-normalized
Router	softmax top-a on q (read) or k (write); balancing loss 10^{-5}
Optimizer	AdamW, $\beta = (0.9, 0.95)$, weight decay 0.05
Learning rate	$4{\times}10^{-4}$, 2,500 warmup steps, cosine to 0
Gradient clip / global batch	1.0 / 128
Training steps	24,476 / 20,046 (objects / scenes; 20,000 rounded up to an epoch)
Loss	pixel MSE

E Language-Modeling Experiments

Figure E1 resolves retrieval by task and validation loss by token position at both model scales.

E.1 Load Balancing

Both balancing mechanisms act on expert *utilization*. For each routing direction $a \in \{\text{read}, \text{write}\}$, let $S^{(a)}$ be the token heads routed within a scope and $c_e^{(a)}$ the number that select expert e. Its utilization is $f_e^{(a)} = c_e^{(a)} / \sum_{e'} c_{e'}^{(a)}$, and its mean routing probability is $\bar{p}_e^{(a)} = |S^{(a)}|^{-1} \sum_{t \in S^{(a)}} p_{t,e}^{(a)}$, where $p_t^{(a)}$ is the softmax of the corresponding logits of (4) over the visible domain of token head t; the superscript is omitted for one direction. With *layer* scope, each layer carries its own penalty or bias vector; with *pool* scope, the statistics cover all L layers that share the pool and one penalty or bias vector serves the whole pool, as in shared expert pools (Huang et al., 2026; Chen et al., 2026).

The *auxiliary loss* (Shazeer et al., 2017; Fedus et al., 2022) adds to the training loss the load-balancing term of Fedus et al. (2022),

$$\mathcal{L}_{\text{bal}} \;=\; \lambda\, E \sum_{a \in \{\text{read},\text{write}\}} \sum_{e=1}^{E} \bar{p}_e^{(a)}\, f_e^{(a)}, \tag{14}$$

with λ the coefficient of Table E1. Its statistics are computed over each data-parallel worker's local batch, read and write are normalized separately, and under layer scope the L per-layer terms are added. The *auxiliary-loss-free bias* (Wang et al., 2024) instead uses the combined counts $c_e =$

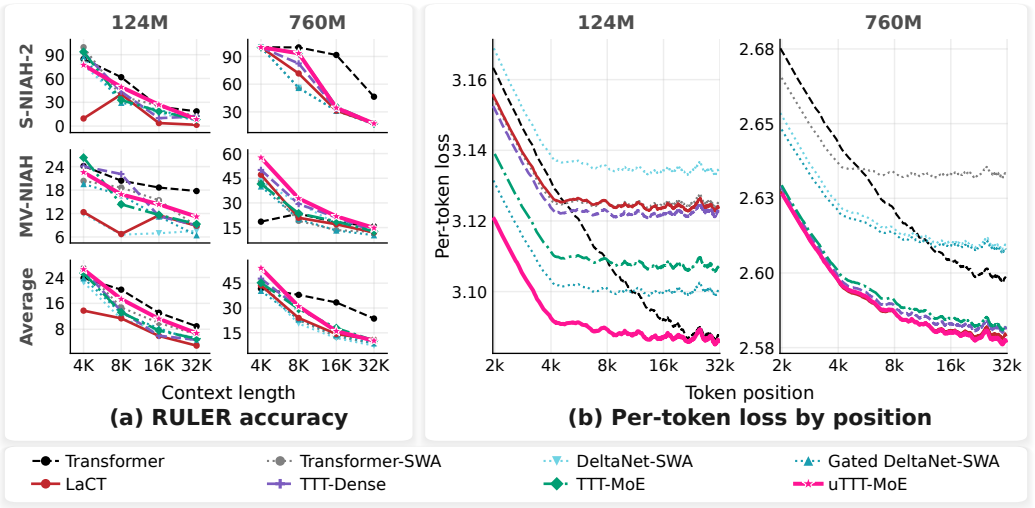


Figure E1: **Retrieval by task and loss by position**, at 124M and 760M. *(a)* RULER accuracy at 4K–32K on the second single-needle task (S-NIAH-2), the multi-value task (MV-NIAH), and, in the last row, the unweighted mean over all thirteen RULER tasks, which is the RULER axis of Figure 3. *(b)* Per-token validation loss on Books3 by position within a 32K-token sequence, on a log-scaled axis with positions below 2K omitted as warm-up.

$c_e^{(\mathrm{read})} + c_e^{(\mathrm{write})}$, summed across data-parallel workers. It keeps a bias b_e per expert in the scope and adds it to the scores that the top-K selection of (4) ranks; after every training step

$$b_e \;\leftarrow\; b_e + u\,\mathrm{sign}\big(\bar{c} - c_e\big), \qquad \bar{c} = \tfrac{1}{E}\textstyle\sum_{e'} c_{e'}, \tag{15}$$

followed by re-centering to zero mean and clipping to $|b_e| \le 2$, with u the *bias-update rate*; inference keeps the trained bias fixed. The gate of a selected expert in (2) and (3) is its routing probability times the *routing-weight scale* γ,

$$g_{\ell,h,i,e} \;=\; \gamma\, p_{t,e} \qquad \text{for } e \in \mathcal{R}_{\ell,h}(z),\; t = (\ell,h,i), \tag{16}$$

with $\gamma = 1$ everywhere except Table E3.

Router utilization. The busiest-expert ratio $E \max_e f_e$ of Section 4.2 is taken from the read routing at the last training step; the write routing gives $6.3\times$ and $1.5\times$ at 124M and $10.8\times$ and $2.1\times$ at 760M for the same unbalanced and reported runs. The unbalanced routers leave 3 of 48 experts below a quarter of an even share at 124M and 74 of 96 at 760M, the reported ones 0 and 3. Counted per layer, layer scope leaves at most 9% of the experts below that share at 124M, whereas pool scope leaves 41–49% at 124M and 89–94% at 760M while the pool as a whole stays even: each layer draws on about half of the pool at 124M and on six to eleven of the 96 experts at 760M. The four-expert layer-private routers stay within $1.22\times$ of an even share at every loss-free rate.

Seeds and sweeps. Figure E2a shows six training seeds of the 124M uTTT-MoE configuration, spanning 14.47 to 16.60 RULER and 3.0753 to 3.0939 in per-token loss. Tables E1 and E2 list every balancing run of the two routed families, Figure E2b plots them, Figure E3 resolves the uTTT-MoE runs by task, and Table E3 varies the routing-weight scale at 760M. The 760M uTTT-MoE configuration keeps the mechanism and scope selected at 124M, with the best of the five bias-update rates tried at 760M. The lead of the unbalanced 760M run comes almost entirely from S-NIAH-1 (91.66 against 57.20) and the second multi-key task (14.06 against 4.00); without S-NIAH-1 it trails the reported run by 2.2 points. Eight private experts per layer, twice the total state of the shared pool, lower the 124M loss of TTT-MoE at every matched setting but do not raise its best RULER.

E.2 A Dense Shared State at 124M

Table E4 reports the dense shared state of Section 3.3, trained at 124M over a ladder of widths r.

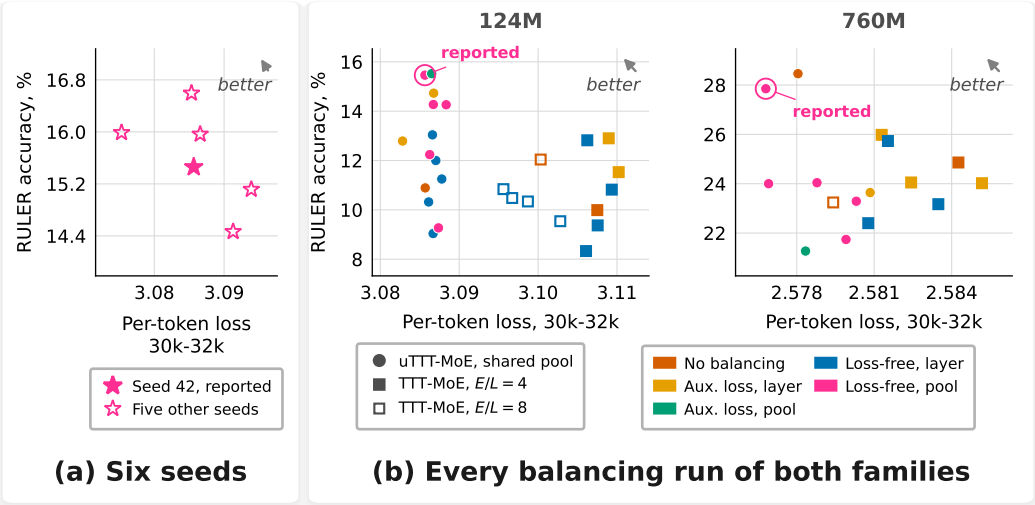


Figure E2: **Training seeds and every balancing run of both routed families**, on the axes of Figure 3. (a) Six training seeds of the 124M uTTT-MoE configuration of Table 1; the filled star is seed 42, the reported run. (b) Every run of Table E1 (uTTT-MoE, circles) and Table E2 (TTT-MoE; filled squares E/L=4, hollow E/L=8) at 124M and 760M. Colour is the balancing mechanism and the scope over which utilization is measured; the layer-private control has a single scope and takes the layer-scope colours. The ringed circle is the configuration reported as uTTT-MoE at that scale.

Table E1: **Every load-balancing run of uTTT-MoE**, at the default routing-weight scale $\gamma = 1$ with a per-layer router; columns as in Table 1. Mechanism, scope and the number in each row label are defined in Appendix E.1. * marks the row reported in Table 1; bold marks the best value in each column within a scale.

Balancing	PTL ↓	RULER ↑	4K	8K	16K	32K
124M, E=48						
No balancing	3.0857	10.89	22.43	10.26	7.33	3.56
Aux. loss 10^{-4}, layer	3.0868	14.73	24.73	16.93	10.94	6.32
Aux. loss 10^{-3}, layer	**3.0828**	12.79	20.05	16.17	8.59	6.37
Aux. loss 10^{-3}, pool	3.0865	**15.52**	26.44	**17.70**	**11.44**	6.49
Loss-free 10^{-5}, layer	3.0866	13.04	25.56	13.12	9.07	4.42
Loss-free 3×10^{-5}, layer	3.0878	11.25	19.88	10.79	9.08	5.25
Loss-free 10^{-4}, layer	3.0870	12.00	23.03	12.47	7.15	5.35
Loss-free 3×10^{-4}, layer	3.0861	10.32	19.79	10.63	6.15	4.71
Loss-free 10^{-3}, layer	3.0867	9.04	18.79	9.02	4.83	3.54
Loss-free 10^{-5}, pool	3.0863	12.24	22.28	14.73	7.61	4.34
Loss-free 3×10^{-5}, pool *	3.0857	15.46	26.50	17.32	11.23	**6.78**
Loss-free 10^{-4}, pool	3.0867	14.27	**26.95**	15.53	9.43	5.19
Loss-free 3×10^{-4}, pool	3.0883	14.26	24.45	16.07	9.80	6.72
Loss-free 10^{-3}, pool	3.0874	9.27	17.02	8.48	6.43	5.15
760M, E=96						
No balancing	2.5780	**28.46**	49.32	30.11	**19.23**	**15.17**
Aux. loss 10^{-3}, layer	2.5808	23.64	48.64	23.27	14.42	8.23
Aux. loss 10^{-3}, pool	2.5783	21.27	38.88	20.36	14.19	11.66
Loss-free 10^{-5}, pool	2.5769	24.00	44.21	26.07	16.47	9.24
Loss-free 3×10^{-5}, pool	2.5803	23.29	45.21	22.55	14.18	11.23
Loss-free 10^{-4}, pool	2.5788	24.04	49.79	24.73	13.85	7.79
Loss-free 3×10^{-4}, pool	2.5799	21.74	40.72	23.11	13.02	10.11
Loss-free 10^{-3}, pool *	**2.5768**	27.85	**54.15**	**31.00**	16.03	10.23

Table E2: **Every layer-private routed run (TTT-MoE)**, the control this paper compares against; columns as in Table 1. E/L is the number of experts per layer, one expert active per token head; the balancing setting reads as in Table E1, without a scope. The tinted row is the one reported in Table 1 at each scale; bold as in Table E1.

Configuration	PTL ↓	RULER ↑	4K	8K	16K	32K
124M						
E/L=4, no balancing	3.1075	9.99	18.53	10.75	6.97	3.70
E/L=4, loss-free 3×10^{-6}	3.1076	9.37	18.68	10.71	4.78	3.33
E/L=4, loss-free 10^{-5}	3.1062	12.82	**25.35**	13.26	7.60	5.08
E/L=4, loss-free 3×10^{-5}	3.1061	8.33	16.50	7.97	4.90	3.96
E/L=4, loss-free 10^{-4}	3.1093	10.82	23.33	9.30	6.33	4.32
E/L=4, aux. loss 10^{-6}	3.1102	11.53	23.15	11.40	6.59	4.97
E/L=4, aux. loss 10^{-5}	3.1090	**12.90**	22.01	**15.64**	7.72	**6.21**
E/L=8, no balancing	3.1003	12.04	24.61	10.83	**7.79**	4.94
E/L=8, loss-free 3×10^{-6}	3.0967	10.48	21.64	9.49	6.41	4.40
E/L=8, loss-free 10^{-5}	3.0987	10.34	20.95	10.23	5.71	4.47
E/L=8, loss-free 3×10^{-5}	**3.0956**	10.84	20.15	10.80	7.74	4.67
E/L=8, loss-free 10^{-4}	3.1028	9.54	17.56	10.55	6.46	3.59
760M						
E/L=4, no balancing	2.5843	24.86	45.57	25.37	16.17	12.33
E/L=4, loss-free 10^{-5}	2.5815	25.73	45.27	**29.44**	17.85	10.35
E/L=4, loss-free 3×10^{-5}	2.5835	23.17	44.05	24.49	14.33	9.83
E/L=4, loss-free 10^{-4}	2.5808	22.40	38.94	26.04	14.64	9.98
E/L=4, aux. loss 10^{-5}	2.5813	**25.98**	45.82	25.34	**18.79**	**13.98**
E/L=4, aux. loss 3×10^{-5}	2.5824	24.05	41.90	24.58	17.27	12.46
E/L=4, aux. loss 10^{-4}	2.5852	24.02	43.52	24.58	15.65	12.35
E/L=8, no balancing	**2.5794**	23.24	**46.29**	22.31	14.50	9.88

Table E3: **The routing-weight scale at 760M.** Pool scope and a per-layer router throughout; γ scales the gates as in (16) and is 1 everywhere else in this paper. Each cell is PTL and RULER (%); a dash marks a combination that was not trained. The γ=1 column is the pool-scope part of the 760M block of Table E1.

	$\gamma = 1$		$\gamma = 2$		$\gamma = 4$	
Balancing	PTL	RULER	PTL	RULER	PTL	RULER
Aux. loss 10^{-4}	—	—	2.5744	23.16	—	—
Aux. loss 5×10^{-4}	—	—	2.5780	23.96	2.5805	24.19
Aux. loss 10^{-3}	2.5783	21.27	2.5783	25.50	2.5791	26.07
Aux. loss 2×10^{-3}	—	—	2.5793	20.57	2.5794	25.56
Aux. loss 5×10^{-3}	—	—	2.5771	25.11	—	—
Loss-free 10^{-5}	2.5769	24.00	—	—	—	—
Loss-free 3×10^{-5}	2.5803	23.29	2.5764	25.57	—	—
Loss-free 10^{-4}	2.5788	24.04	2.5735	25.21	2.5752	22.37
Loss-free 3×10^{-4}	2.5799	21.74	2.5757	25.68	—	—
Loss-free 10^{-3}	2.5768	27.85	—	—	—	—

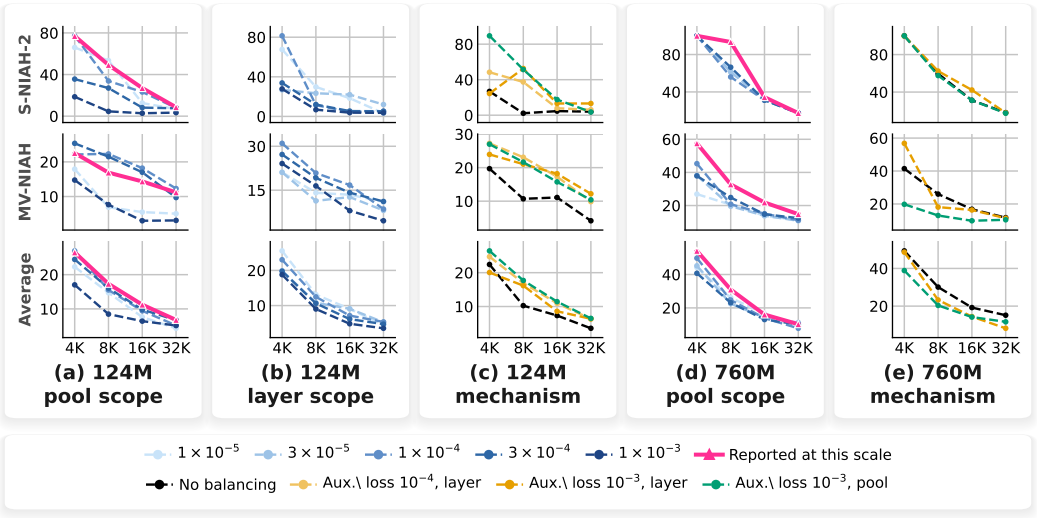


Figure E3: **Every run of Table E1, resolved by task.** Cards (a), (b) and (d) sweep the loss-free bias-update rate at 124M pool scope, 124M layer scope and 760M pool scope; cards (c) and (e) draw the unbalanced and auxiliary-loss runs at 124M and 760M. Each card holds the two tasks of Figure E1(a) and the unweighted mean over all thirteen RULER tasks, whose four points are the 4K–32K columns of Table E1; the cards share their rows, so a task reads across the figure. Rates are ordinal and drawn light to dark; the pink line is the configuration reported at that scale, present only in the two pool-scope sweeps. Each panel carries its own accuracy range.

Table E4: **A dense shared state at 124M.** The shared state is r times the width of one layer-private state, so r=1 matches TTT-Dense in per-token fast-weight compute and r=12, the depth of the model, would match it in size; columns as in Table 1; the TTT-Dense row repeats Table 1.

Model	State (M)	PTL ↓	RULER ↑	4K	8K	16K	32K
uTTT-Dense, r=1 (FLOPs-matched)	0.44	3.1290	10.64	20.27	12.23	6.46	3.61
uTTT-Dense, r=2	0.88	3.1232	13.20	23.57	14.66	9.79	4.78
uTTT-Dense, r=4	1.77	3.1249	13.41	21.65	15.12	10.69	6.20
uTTT-Dense, r=6	2.65	3.1165	13.63	23.60	16.01	9.92	5.01
TTT-Dense (Table 1)	5.31	3.1219	12.65	26.31	13.58	6.00	4.72

F LENGTH EXTRAPOLATION

This appendix evaluates seven 124M models beyond the 32K training length, per-token loss to 128K and retrieval to 64K (Table F2): five trained at 32K and two attention references trained at 64K (Appendix D).

Per-position loss confounds context length with document depth: later tokens may be easier even without access to more history. We therefore pair the full-document curves (Figure F1; 3,000 Books3 documents of at least 256K tokens, each read from its first token to 131,072) with two controls on the same documents (Table F1): a *fixed-target ablation*, which scores the same 8,192 tokens at positions 120–128K while the visible prefix grows from 4K to 120K, on a 512-document subset; and *reset segments*, which evaluate the same 128K as four consecutive 32K pieces, each from position 0. Means exclude each row's final position.

In the fixed-target ablation (Table F1, left), the 32K-trained Transformer-SWA stays at 3.222 nats at the reported precision from a 4K to a 120K prefix, so it serves as a position-matched reference. With context reset every 32K (Table F1, right), every model still improves by 0.010–0.015 nats from the first segment to the fourth: later text is easier by itself, so we compare models against this reference rather than by raw loss.

Table F1: **Fixed-target ablation and reset segments**, loss in nats. Left: the same 8,192 target tokens as the visible prefix grows; bold marks each row's minimum before rounding. Right: the four consecutive 32K pieces of the same 128K. Transformer-SWA has a 4K attention window.

	Fixed-target ablation: visible prefix							Reset segments			
Model	4K	8K	16K	32K	48K	64K	120K	Seg. 1	Seg. 2	Seg. 3	Seg. 4
Transformer-SWA (32K)	3.222	**3.222**	**3.222**	**3.222**	**3.222**	**3.222**	**3.222**	3.224	3.224	3.218	3.214
Transformer-SWA (64K)	3.220	**3.220**	3.220	3.220	3.220	**3.220**	3.220	3.223	3.222	3.216	3.212
Transformer (32K)	3.211	3.198	**3.188**	3.329	4.920	5.692	5.838	3.205	3.201	3.194	3.190
Transformer (64K)	3.226	3.212	3.201	**3.193**	3.194	3.208	5.563	3.218	3.215	3.208	3.204
LaCT	3.215	3.214	**3.213**	3.213	3.216	3.219	3.230	3.216	3.216	3.209	3.205
TTT-MoE	3.189	3.188	3.187	**3.187**	3.188	3.188	3.189	3.190	3.190	3.184	3.180
uTTT-MoE	3.189	3.187	3.186	**3.185**	3.186	3.187	3.188	3.189	3.189	3.183	3.179

Figure F1(b) shows each model's loss relative to the reference at the same positions, $\Delta(p) = L_{\text{model}}(p) - L_{\text{SWA}}(p)$. Within 32K, uTTT-MoE has the lowest curve; the expert pools sit about 0.035 nats below SWA and LaCT about 0.010, while the two SWA references stay close. Both expert pools keep this advantage to 128K, whereas LaCT crosses zero near 85K and full attention deteriorates sharply beyond its trained length, later for the 64K-trained reference.

Table F2: **RULER accuracy (%) by evaluation length**, 500 samples per task per length, averaged over the 13 tasks of Figure E1. Bold marks column maxima at the displayed precision. †: beyond the trained length. Model configurations are given in Appendix D.

Model	4K	8K	16K	32K	64K
Transformer-SWA, 4K window (32K)	29.1	15.6	9.8	6.0	3.4†
Transformer-SWA, 4K window (64K)	28.5	16.0	9.5	6.0	3.8
Transformer (32K)	**30.9**	**26.4**	**23.0**	**13.8**	1.4†
Transformer (64K)	20.5	10.7	11.4	8.4	**5.8**
LaCT	23.6	14.8	9.3	5.2	3.3†
TTT-MoE	28.7	13.3	6.4	4.2	3.3†
uTTT-MoE	**30.9**	17.7	9.8	5.9	3.7†

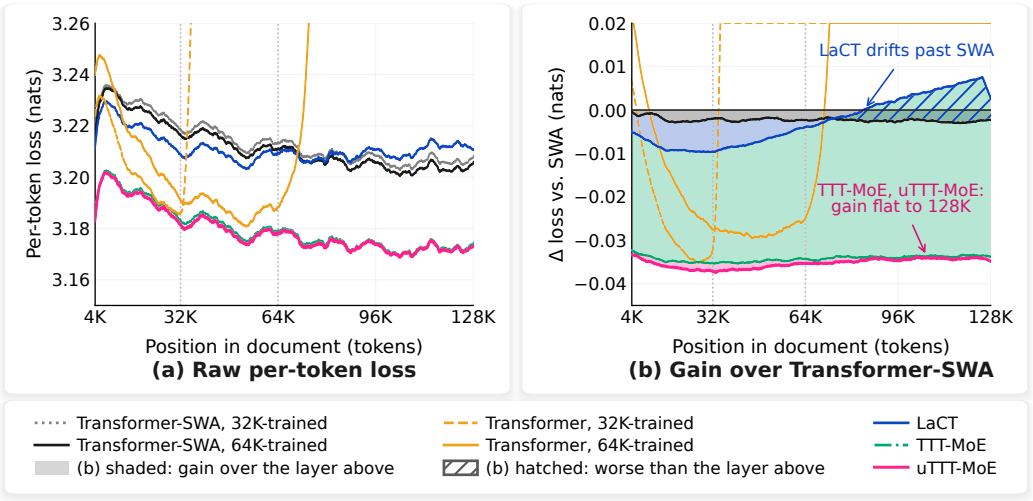


Figure F1: **Per-token loss along** 3,000 **Books3 documents (positions** 4K**–**128K**)**, smoothed over 6,144 tokens. *(a)* Loss of the seven models. *(b)* Loss relative to the 32K-trained Transformer-SWA with a 4K attention window. Shaded bands show successive loss differences in the order SWA (64K), LaCT, TTT-MoE, uTTT-MoE, starting from zero; hatching marks a reversal of the ordering. Full-attention models are lines only, clipped at the top. Dotted lines mark the 32K and 64K training lengths; LaCT and both expert pools are 32K-trained.

G Efficiency measurements

Figure 5a reports median per-GPU throughput over steps 50–100 of a 100-step run; panel c computes GPU-hours using each run's throughput at its training batch. Forward FLOPs per token differ by at most 0.6% across the three models, and all use blockwise recomputation to reduce training memory. The schedules and the execution order they impose are defined in Appendix C.3; this appendix measures their cost.

At 124M, the throughput advantage depends on hardware and batch size. On H100, uTTT-MoE is 1.14–1.16× faster than TTT-Dense across the tested batches. In the eight-GPU training runs at 124M and 760M with batch 1 per GPU, peak memory is 1.29–1.37× TTT-Dense's. Increasing the shared pool from 12 to 96 experts at 124M changes training throughput by less than 2% and adds 1.7 GiB of peak memory (Figure G1b). With top-K fixed, the larger pool expands accessible fast-weight state while keeping prefill throughput and per-token decoding latency nearly unchanged at batch 1 and 32K context on one A100.

TTT-MoE and uTTT-MoE use the same fused Triton kernels. Table G1 ablates these kernels in uTTT-MoE by replacing each group with split references of the same arithmetic. Together they improve throughput by 15% at 124M and 18% at 760M, chiefly through routing permutation and, at 760M, the backward pass. With all fusion disabled, uTTT-MoE matches TTT-Dense at 124M and is 6% slower at 760M in this single-GPU setting. It remains 19% and 15% faster, respectively, than TTT-MoE with fusion.

At 760M on one A100, uTTT-MoE prefills at batch 1 as fast as TTT-Dense and 1.3× faster than TTT-MoE at the tested 16K and 32K prompt lengths (Figure G1c). Increasing batch size from 1 to 4 brings little improvement in prefill throughput at these lengths. It decodes at TTT-MoE's speed, 0.72–0.74× TTT-Dense's across the tested 4K–128K contexts. At batch 1, time to first token is 29–33% lower than TTT-MoE's at 32K and 128K. Chunk-boundary pauses, measured as the maximum boundary-token step latency per run, are 26–27% shorter at the tested 4K, 32K and 128K contexts. Both latency metrics are comparable to TTT-Dense in these settings; comparisons use medians of three runs.

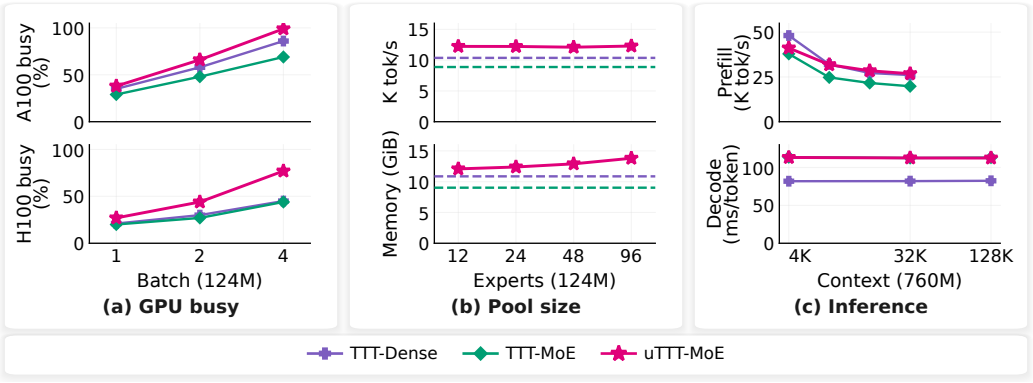


Figure G1: **Efficiency beyond Figure 5.** *(a)* GPU busy time at 124M against batch per GPU, on one node of eight A100s (top) or H100s (bottom). *(b)* uTTT-MoE training throughput (top) and peak memory (bottom) against shared-pool size at 124M on one A100, with TTT-Dense and TTT-MoE in the same setting dashed. *(c)* 760M inference against context length: batch-1 prefill throughput (top, one A100, 4K–32K contexts) and per-token decoding latency (bottom, one A100; TTT-MoE and uTTT-MoE coincide).

Table G1: **Fused-kernel ablation of uTTT-MoE training** at batch 1×32K on one A100. Training tokens per second per GPU, median of three repeats, and the speed-up the group gives; peak memory is the same in every row.

	124M		760M	
Kernels switched off	tok/s	speed-up	tok/s	speed-up
none	12,071	–	5,542	–
all	10,533	1.15	4,710	1.18
routing permutation of k, v and learning rates	11,282	1.07	5,088	1.09
grouped GEMMs of the outer-gradient backward	11,816	1.02	5,158	1.07
pointwise second-order backward	11,950	1.01	5,162	1.07
inner gradient of the fast-weight write	12,033	1.00	5,478	1.01
fast-weight read	12,197	0.99	5,498	1.01

H Additional Novel View Synthesis Comparisons

This appendix holds the novel-view-synthesis results that Section 5 refers to but does not display: the modules of Figure 6 on DL3DV, the remaining modules on both datasets, every configuration with its intervals, and the comparison of the two write schedules. Protocol and notation are those of Section 5.1 and Figure 6.

H.1 Sharing and Compute on Scenes

Figure H1 repeats the four modules of Figure 6 on DL3DV scenes. Row B carries over: activating more experts raises PSNR on scenes as on objects, by 0.54 dB over all views from a=1 to a=8 with non-overlapping intervals, and widening the dense state raises it by 0.31 dB. The differences among the four sharing levels in row A are not significant on DL3DV. The sharing conclusion therefore rests on objects, whose test set is seven times larger; the compute conclusion holds on both datasets.

H.2 Total Experts and Pool Size

Rows A and B hold the total at 64 experts. MoE A′ and B′ (Figures H2 and H3) repeat both sweeps at 32 experts in total, with TTT-MoE-e4 as the layer-private end point, to check that the orderings are properties of sharing and of compute rather than of one total. Both hold: quality rises at every merge on Objaverse/GSO and the activation sweep is monotone on both datasets, with non-overlapping intervals between the end points of both sweeps on objects and of the activation sweep on scenes.

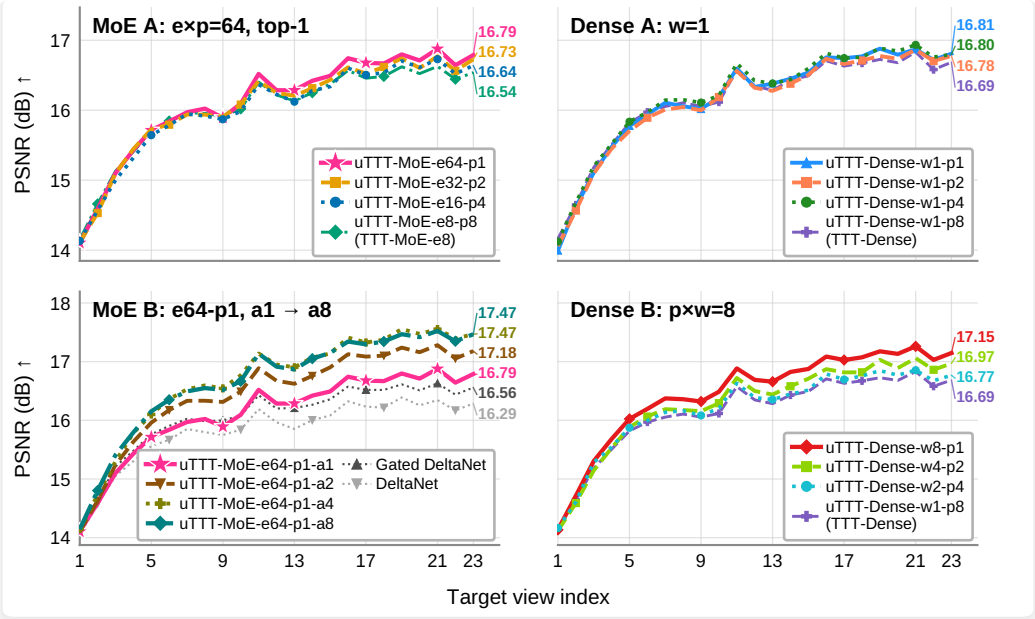


Figure H1: **The four modules of Figure 6 on DL3DV**; conventions as there.

Module C separates sharing from the wider routing choice that comes with it: sharing is fixed at $p{=}1$, per-layer compute at $a{=}1$, and only the pool size varies, from 8 to 64 experts. If choice or stored state were what mattered, quality would rise with the pool; the four pools instead finish within 0.25 dB of one another on objects, the smallest the best, and within 0.16 dB on scenes, and on Objaverse/GSO the 8-expert pool leads TTT-MoE-e8 over all views with non-overlapping intervals at one eighth of its experts and the same eight candidates per token head (Section 5.2). Row A therefore measures that the state a layer reads was written by every layer, not that it has more to choose from. The dense state has no such control, since widening it raises per-layer compute together with state; the width sweep of module C is the dense counterpart of row B and behaves like one, width 8 leading on both datasets while the intermediate widths are not ordered on objects.

H.3 Every Configuration

Tables H1 and H2 list every fast-weight configuration and the reference models on each dataset in one order: the layer-private pools, the universal pool at $a{=}1$, the depth partitions at fixed total, the activation sweeps, the universal dense state and the dense partitions.

H.4 Write Schedules

We compare the 64-expert, $a{=}1$ shared pool under the two schedules of Appendix C.3 using full-test results on both datasets (Table H3). At target view 23, the aggregated schedule has about 0.1 dB lower PSNR on Objaverse/GSO and 0.01 dB lower on DL3DV, trains 2.1% faster and uses 26.3% less peak memory on one H100; the canonical sweeps use it.

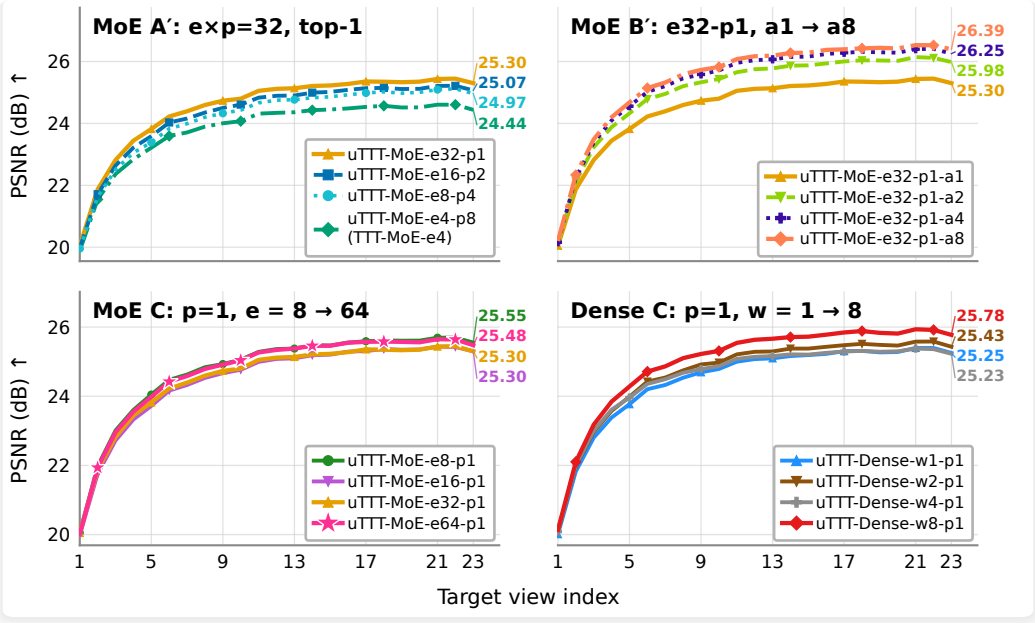


Figure H2: **The remaining modules on Objaverse/GSO**; conventions as Figure 6. MoE A′ and B′ repeat rows A and B at 32 experts in total, with TTT-MoE-e4 as the p=8 endpoint. C sweeps the pool size under full sharing (p=1): per-layer compute is fixed for the routed pool (a=1) and scales with w for the dense state.

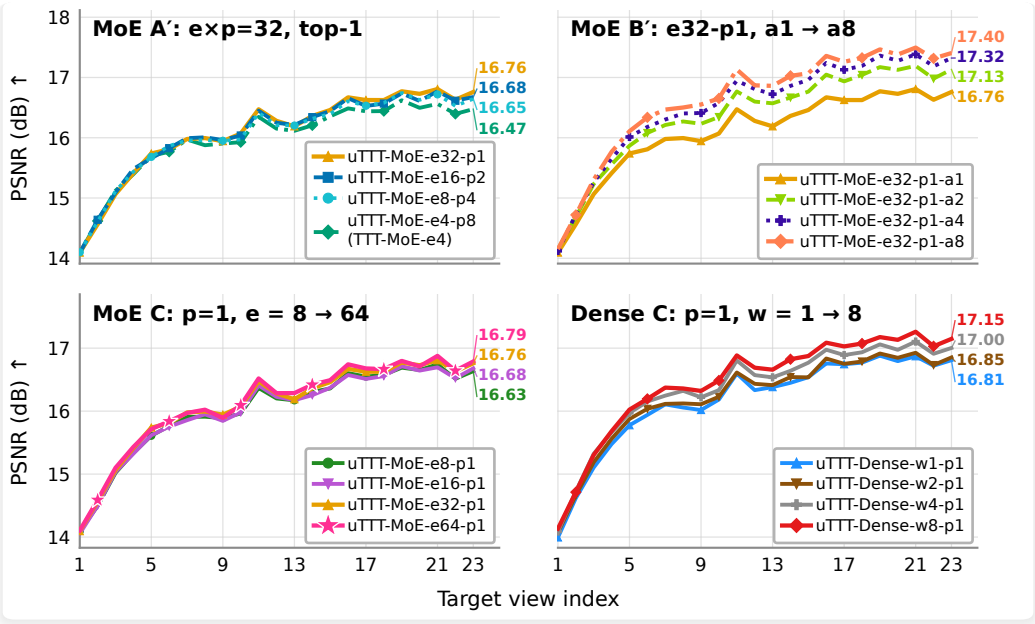


Figure H3: **The modules of Figure H2 on DL3DV**; conventions as Figure 6.

Table H1: **Every fast-weight configuration on Objaverse/GSO** (n=1,019 objects). Cells: mean over objects of each object's mean over target views 1–23, with its 95% bootstrap interval over objects; the second PSNR column is view 23 alone. Bold: best per column among fast-weight models; †full-attention reference, not bolded.

Model	PSNR ↑	View 23 PSNR ↑	SSIM ↑	LPIPS ↓
Layer-private pools (p=8)				
uTTT-Dense-w1-p8 (TTT-Dense)	23.99 [23.81, 24.16]	24.49 [24.29, 24.69]	0.852 [0.847, 0.856]	0.185 [0.180, 0.189]
uTTT-MoE-e4-p8 (TTT-MoE-e4)	23.80 [23.62, 23.97]	24.44 [24.24, 24.64]	0.849 [0.845, 0.854]	0.188 [0.183, 0.193]
uTTT-MoE-e8-p8 (TTT-MoE-e8)	23.90 [23.73, 24.08]	24.56 [24.36, 24.77]	0.850 [0.845, 0.855]	0.186 [0.181, 0.191]
MoE, universal pool, top-1 (C)				
uTTT-MoE-e8-p1	24.72 [24.54, 24.90]	25.55 [25.34, 25.77]	0.861 [0.856, 0.865]	0.172 [0.168, 0.177]
uTTT-MoE-e16-p1	24.45 [24.27, 24.63]	25.30 [25.09, 25.50]	0.857 [0.853, 0.862]	0.176 [0.172, 0.181]
uTTT-MoE-e32-p1	24.49 [24.32, 24.67]	25.30 [25.10, 25.51]	0.858 [0.854, 0.863]	0.175 [0.170, 0.179]
uTTT-MoE-e64-p1	24.68 [24.51, 24.86]	25.48 [25.27, 25.68]	0.860 [0.855, 0.864]	0.173 [0.168, 0.177]
MoE, depth partitions at fixed total (A, A′)				
uTTT-MoE-e32-p2	24.20 [24.03, 24.37]	24.98 [24.78, 25.18]	0.854 [0.850, 0.859]	0.181 [0.177, 0.186]
uTTT-MoE-e16-p4	24.11 [23.94, 24.29]	24.89 [24.69, 25.10]	0.854 [0.849, 0.858]	0.182 [0.177, 0.186]
uTTT-MoE-e16-p2	24.28 [24.11, 24.46]	25.07 [24.87, 25.27]	0.855 [0.850, 0.859]	0.180 [0.176, 0.185]
uTTT-MoE-e8-p4	24.14 [23.97, 24.32]	24.97 [24.76, 25.17]	0.854 [0.849, 0.858]	0.181 [0.177, 0.186]
MoE, active width (B, B′)				
uTTT-MoE-e64-p1-a2	25.18 [25.00, 25.36]	26.12 [25.91, 26.33]	0.867 [0.862, 0.871]	0.164 [0.159, 0.168]
uTTT-MoE-e64-p1-a4	25.38 [25.20, 25.56]	26.35 [26.14, 26.56]	0.869 [0.864, 0.873]	0.161 [0.156, 0.165]
uTTT-MoE-e64-p1-a8	**25.56** [25.38, 25.74]	**26.55** [26.34, 26.76]	**0.871** [0.867, 0.876]	**0.158** [0.154, 0.163]
uTTT-MoE-e32-p1-a2	25.07 [24.89, 25.25]	25.98 [25.77, 26.20]	0.865 [0.861, 0.870]	0.165 [0.161, 0.170]
uTTT-MoE-e32-p1-a4	25.32 [25.14, 25.50]	26.25 [26.05, 26.47]	0.868 [0.863, 0.872]	0.163 [0.158, 0.167]
uTTT-MoE-e32-p1-a8	25.44 [25.26, 25.62]	26.39 [26.18, 26.61]	0.869 [0.865, 0.873]	0.160 [0.156, 0.165]
Dense, universal state (C)				
uTTT-Dense-w1-p1	24.45 [24.27, 24.62]	25.25 [25.05, 25.46]	0.858 [0.853, 0.862]	0.176 [0.172, 0.181]
uTTT-Dense-w2-p1	24.64 [24.46, 24.82]	25.43 [25.22, 25.64]	0.860 [0.855, 0.864]	0.174 [0.169, 0.178]
uTTT-Dense-w4-p1	24.53 [24.34, 24.71]	25.23 [25.02, 25.44]	0.858 [0.854, 0.863]	0.175 [0.170, 0.179]
uTTT-Dense-w8-p1	24.95 [24.77, 25.13]	25.78 [25.57, 25.99]	0.864 [0.859, 0.868]	0.167 [0.163, 0.172]
Dense, depth partitions (B, A)				
uTTT-Dense-w4-p2	24.75 [24.58, 24.93]	25.56 [25.36, 25.77]	0.861 [0.857, 0.866]	0.172 [0.167, 0.176]
uTTT-Dense-w2-p4	24.53 [24.35, 24.71]	25.30 [25.09, 25.51]	0.859 [0.854, 0.863]	0.174 [0.169, 0.179]
uTTT-Dense-w1-p2	24.35 [24.17, 24.53]	25.14 [24.93, 25.34]	0.856 [0.852, 0.861]	0.179 [0.175, 0.184]
uTTT-Dense-w1-p4	24.37 [24.20, 24.55]	25.19 [24.98, 25.40]	0.858 [0.853, 0.862]	0.177 [0.172, 0.182]
Baseline				
Transformer†	25.36 [25.19, 25.54]	26.54 [26.33, 26.74]	0.870 [0.865, 0.874]	0.162 [0.158, 0.166]
Gated DeltaNet	24.32 [24.14, 24.50]	25.01 [24.80, 25.21]	0.856 [0.851, 0.860]	0.178 [0.173, 0.182]
DeltaNet	23.94 [23.76, 24.12]	24.55 [24.34, 24.75]	0.851 [0.846, 0.856]	0.185 [0.180, 0.189]

Table H2: **Every fast-weight configuration on DL3DV** (n=140 test scenes); format as Table H1, with means and bootstrap intervals computed over scenes.

Model	PSNR ↑	View 23 PSNR ↑	SSIM ↑	LPIPS ↓
Layer-private pools (p=8)				
uTTT-Dense-w1-p8 (TTT-Dense)	16.14 [15.91, 16.39]	16.69 [16.35, 17.03]	0.383 [0.365, 0.402]	0.681 [0.673, 0.690]
uTTT-MoE-e4-p8 (TTT-MoE-e4)	15.98 [15.74, 16.22]	16.47 [16.14, 16.81]	0.381 [0.362, 0.399]	0.691 [0.682, 0.699]
uTTT-MoE-e8-p8 (TTT-MoE-e8)	16.02 [15.78, 16.27]	16.54 [16.20, 16.89]	0.379 [0.361, 0.397]	0.688 [0.679, 0.696]
MoE, universal pool, top-1 (C)				
uTTT-MoE-e8-p1	16.02 [15.78, 16.27]	16.63 [16.30, 16.98]	0.382 [0.364, 0.401]	0.686 [0.677, 0.694]
uTTT-MoE-e16-p1	16.01 [15.77, 16.26]	16.68 [16.35, 17.02]	0.382 [0.364, 0.401]	0.686 [0.678, 0.695]
uTTT-MoE-e32-p1	16.09 [15.85, 16.34]	16.76 [16.41, 17.11]	0.384 [0.366, 0.403]	0.683 [0.674, 0.692]
uTTT-MoE-e64-p1	16.12 [15.88, 16.37]	16.79 [16.44, 17.14]	0.385 [0.367, 0.403]	0.682 [0.673, 0.690]
MoE, depth partitions at fixed total (A, A′)				
uTTT-MoE-e32-p2	16.06 [15.83, 16.31]	16.73 [16.39, 17.07]	0.383 [0.365, 0.401]	0.685 [0.676, 0.693]
uTTT-MoE-e16-p4	16.02 [15.78, 16.27]	16.64 [16.30, 16.99]	0.382 [0.364, 0.401]	0.688 [0.679, 0.696]
uTTT-MoE-e16-p2	16.07 [15.84, 16.32]	16.68 [16.34, 17.02]	0.382 [0.363, 0.400]	0.685 [0.677, 0.694]
uTTT-MoE-e8-p4	16.05 [15.81, 16.31]	16.65 [16.31, 16.99]	0.382 [0.363, 0.400]	0.687 [0.679, 0.696]
MoE, active width (B, B′)				
uTTT-MoE-e64-p1-a2	16.45 [16.20, 16.71]	17.18 [16.82, 17.54]	0.394 [0.376, 0.413]	0.668 [0.659, 0.678]
uTTT-MoE-e64-p1-a4	**16.68** [16.43, 16.94]	17.47 [17.09, 17.84]	**0.401** [0.383, 0.420]	**0.659** [0.650, 0.668]
uTTT-MoE-e64-p1-a8	16.66 [16.41, 16.92]	**17.47** [17.10, 17.84]	0.400 [0.382, 0.419]	0.660 [0.651, 0.669]
uTTT-MoE-e32-p1-a2	16.38 [16.13, 16.63]	17.13 [16.78, 17.49]	0.393 [0.375, 0.412]	0.674 [0.665, 0.683]
uTTT-MoE-e32-p1-a4	16.52 [16.27, 16.79]	17.32 [16.95, 17.70]	0.397 [0.378, 0.415]	0.666 [0.657, 0.675]
uTTT-MoE-e32-p1-a8	16.63 [16.38, 16.89]	17.40 [17.04, 17.78]	0.399 [0.381, 0.418]	0.661 [0.652, 0.671]
Dense, universal state (C)				
uTTT-Dense-w1-p1	16.17 [15.93, 16.43]	16.81 [16.47, 17.16]	0.386 [0.368, 0.404]	0.683 [0.674, 0.692]
uTTT-Dense-w2-p1	16.22 [15.98, 16.48]	16.85 [16.51, 17.20]	0.387 [0.369, 0.406]	0.680 [0.671, 0.689]
uTTT-Dense-w4-p1	16.36 [16.11, 16.62]	17.00 [16.66, 17.36]	0.392 [0.373, 0.410]	0.676 [0.666, 0.685]
uTTT-Dense-w8-p1	16.45 [16.21, 16.71]	17.15 [16.80, 17.50]	0.394 [0.376, 0.413]	0.669 [0.660, 0.678]
Dense, depth partitions (B, A)				
uTTT-Dense-w4-p2	16.28 [16.04, 16.53]	16.97 [16.62, 17.33]	0.390 [0.372, 0.409]	0.676 [0.666, 0.685]
uTTT-Dense-w2-p4	16.20 [15.96, 16.45]	16.77 [16.43, 17.11]	0.386 [0.367, 0.404]	0.679 [0.670, 0.688]
uTTT-Dense-w1-p2	16.13 [15.89, 16.39]	16.78 [16.44, 17.12]	0.385 [0.367, 0.403]	0.683 [0.674, 0.692]
uTTT-Dense-w1-p4	16.22 [15.98, 16.47]	16.80 [16.46, 17.15]	0.385 [0.367, 0.404]	0.680 [0.671, 0.688]
Baseline				
Transformer†	17.41 [17.14, 17.68]	18.55 [18.12, 18.99]	0.426 [0.408, 0.445]	0.626 [0.617, 0.636]
Gated DeltaNet	16.05 [15.81, 16.29]	16.56 [16.23, 16.89]	0.382 [0.364, 0.400]	0.681 [0.672, 0.690]
DeltaNet	15.82 [15.59, 16.06]	16.29 [15.96, 16.62]	0.376 [0.358, 0.395]	0.689 [0.680, 0.697]

Table H3: **Measured cost and final quality of the two write schedules** at 64 experts, a=1. Aggregated-schedule quality is taken from Tables H1 and H2. View-23 PSNR is the final target view; *min/1k steps* is wall-clock minutes per thousand optimizer steps. Cost measurements use one H100 80GB with the same 256×256 shape, 8 layers, 12 views, batch 32, and no activation checkpointing. Bold marks the best value per column.

Schedule	GSO V23 PSNR ↑	DL3DV V23 PSNR ↑	k patch tok/s/GPU ↑	min/1k steps ↓	Peak GiB ↓
Sequential schedule	**25.58**	**16.80**	100.2	119.9	40.75
Aggregated schedule	25.48	16.79	**102.3** (+2.1%)	**117.5** (-2.1%)	**30.04** (-26.3%)

I Analysis of Shared Memory

We analyze the shared memory in three steps. Appendix I.1 derives the mechanism: with a shared state, a read can depend on the writes of every layer in earlier write-enabled chunks, deeper layers included (Proposition 1). Appendix I.2 specifies the experiments of Figure 4 that test these dependencies in trained models, and Appendix I.3 shows a behavior that sharing enables: repeated observations of a surface preferentially reuse the same experts across views and depths.

I.1 Mechanism: recurrence and cross-depth reach

Both write schedules of Appendix C.3 read the chunk-start state throughout each chunk. Once the state is shared, depth joins the time axis through a joint fast-weight state recurrence over (chunk, layer) pairs (Figure 2a shows the aggregated case). Indexing the pairs in execution order by $t = (c-1)L + \ell$, the sequential schedule becomes

$$W^{(t)} \;=\; \mathrm{Update}\left(W^{(t-1)},\, G_t\right), \tag{17}$$

a *single* recurrence of $(N-1)L$ steps over the whole fast-weight memory in language modeling; the chunk-start state is read throughout each chunk. Step t runs layer $\ell(t) = 1 + ((t-1) \bmod L)$ with that layer's own slow weights; tying them across layers would give a Universal-Transformer-style recurrence (Dehghani et al., 2019), which we do not require. Depth thus adds test-time optimization: in each chunk the shared memory takes L steps under the sequential schedule, or one step that combines the contributions of every depth under the aggregated schedule, whereas a layer-private model advances each of its L memories by one step. A write by layer r in chunk s can be read in the next chunk by any layer, including those below r; this opens the top-down route of Section 3.4 alongside the bottom-up one. Layer-private TTT is a strict special case: at $p = L$, each G_t updates one private state block as in (1); intermediate p gives p group blocks. With producer inputs fixed, the direct state-transition Jacobian is block diagonal under either schedule. Query and write producers remain coupled across layers through the residual stream.

Full and memory-chain derivatives. Let $u_{s,r}$ be the write inputs (keys, values and step sizes) of layer r in chunk s, and $o_{c,\ell}$ the fast-weight read of layer ℓ in chunk c. Nonzero-reach statements below assume that chunk s is write-enabled. We write $\partial o_{c,\ell}/\partial u_{s,r}$ for the *full* derivative: the response of the read to an independent perturbation added at the write inputs, with the graph that produces the queries and the other write inputs left connected. Figure 2b draws a restricted part of it, the *memory-chain* derivative, which holds the queries and predicted momentum coefficients at their forward values and treats the write inputs, step sizes included, as independent variables; the momentum buffers remain recurrent and differentiable. Let $S^{(j)}_{\mathrm{mem}}$ collect the fast weights, master accumulators and momentum buffers after chunk j, and $J_j = \partial S^{(j)}_{\mathrm{mem}}/\partial S^{(j-1)}_{\mathrm{mem}}$ with producer inputs fixed. Since $u_{s,r}$ enters the state through its contribution $G_{s,r}$ in (12)–(13), for write-enabled $s < c$

$$D^{\mathrm{mem}}_{u_{s,r}} o_{c,\ell} \;=\; \frac{\partial o_{c,\ell}}{\partial S^{(c-1)}_{\mathrm{mem}}} \left(J_{c-1} \cdots J_{s+1}\right) \frac{\partial S^{(s)}_{\mathrm{mem}}}{\partial G_{s,r}} \frac{\partial G_{s,r}}{\partial u_{s,r}}. \tag{18}$$

The product runs from the latest update to the earliest and is the identity when $c = s+1$. Every state Jacobian keeps the complete update dependence, including the inner gradient, momentum and renormalization, and the final factors describe how $G_{s,r}$ enters the boundary update: in one step under (13), or at step r followed by the remaining layer steps under (12). It is zero for $s \geq c$, because a chunk never reads its own writes, and it excludes paths that leave the memory chain through a hidden state, a query or a predicted coefficient, including paths that later re-enter the chain through another write. Unless the memory-chain derivative is named, the proposition below and the probe of Appendix I.2 refer to the full derivative, which includes these paths. A historical write of a shallower layer can therefore receive gradient in a layer-private model even though its node is hollow in Figure 2b; the figure depicts potential paths, whose existence does not establish their magnitude or their contribution to performance. Keeping the boundary update in the graph is backpropagation through time over chunks (Section 3.4), and (18) is one of its terms.

Proposition 1 (Reach of a read) (i) *In every model, $\partial o_{c,\ell}/\partial u_{s,r} = 0$ for $s \geq c$; the language-modeling sequence omits the final-chunk write.* (ii) *With private memories ($p = L$), the derivative is*

identically zero for deeper writers, $r > \ell$; shallower writers can reach the read through paths that also traverse the residual stream. (iii) *With a universal pool ($p = 1$) and write-enabled $s < c$, under either schedule, the memory-chain derivative* (18) *is generically nonzero for a dense pool, and for a routed pool when the writer and the reader select a common expert. Pooled momentum, when used, can add paths through its input-dependent coefficient.*

Proof of Proposition 1. A model with L layers processes N chunks. The memory state carried across a chunk boundary consists of fast weights W, master accumulators U and, where used, momentum buffers M, updated as specified in Appendix C.2; a layer-private model carries L blocks (W_r, U_r, M_r), and a universal model carries one pool. Within chunk c every layer reads $W^{(c-1)}$. The write inputs $u_{c,r} = (k, v, \eta)$ of layer r are computed from the layer's input hidden states and enter the computation only through $G_{c,r}$; the momentum coefficient, when used, is predicted from each layer's inputs and averaged over the layers of a pool. Information crosses a chunk boundary only through the memory state and, in language modeling, through each layer's sliding-window attention cache; within a chunk it moves upward through the residual stream.

(i) The write inputs $u_{s,r}$ have a single child, $G_{s,r}$, which enters the state at the boundary after chunk s. A read in chunk $c \le s$ is a function of $W^{(c-1)}$, with $c-1 < s$, and of chunk-c hidden states that do not involve $G_{s,r}$; hence $\partial o_{c,\ell}/\partial u_{s,r} = 0$ for $s \ge c$. In the language-modeling sequence, no boundary update is computed after the last chunk, so $u_{N,r}$ has no child at all.

(ii) Layer-private, $p = L$. Along every edge of the forward graph the layer index is non-decreasing: within a chunk the residual stream carries the output of layer j to the layers above it, and across chunks the memory block (W_j, U_j, M_j) and the attention cache of layer j are written and read by layer j alone. A path from $u_{s,r}$ therefore never reaches a quantity of a layer $\ell < r$, and $\partial o_{c,\ell}/\partial u_{s,r} = 0$ for $r > \ell$ whatever the weights and the input. For $r < \ell$ the memory-chain path is closed as well, since $G_{s,r}$ does not enter the update of W_ℓ, but the residual stream is open: $u_{s,r}$ enters $W_r^{(s)}$, and its effect can persist in layer r's memory through subsequent updates or read-only chunks. In any later chunk $c > s$, layer r can read this state and pass the effect up the residual stream to $o_{c,\ell}$. At an intervening write-enabled chunk, the effect may also enter layer ℓ's own memory through its write inputs. These derivatives are generically nonzero.

(iii) Universal, $p = 1$. Under (13), $W^{(s)} = \mathrm{Update}\left(W^{(s-1)}, \sum_r G_{s,r}\right)$ depends on every $G_{s,r}$; under (12), $G_{s,r}$ enters step r, and the later steps carry its effect to $W^{(s)}$. Because $W^{(s')}$ is a function of $W^{(s)}$ for $s' > s$, $W^{(c-1)}$ is a function of $u_{s,r}$ for every layer r and every write-enabled chunk $s < c$. A layer reading a dense pool uses all H head blocks, while each head reads its corresponding block; neither uses expert routing. The layer-level read-state Jacobian in (18) therefore has support across these blocks, and the memory-chain product is nonzero for generic weights and inputs. In a routed pool, $G_{s,r}$ touches the experts that layer r's write router selects in chunk s, and $o_{c,\ell}$ reads the experts that layer ℓ's read router selects, so the memory-chain path is open exactly when the two selections share an expert. Cross-depth memory reuse does not require momentum. When momentum is used, its pooled coefficient depends on the input and adds a path through the coefficient predictor. The coefficient is predicted from every layer's inputs, averaged over the layers sharing the pool and applied to every expert's momentum buffer at the boundary. A write of layer r in chunk s that changes the inputs of the layers above r in chunk $s+1$ therefore changes the coefficient at boundary $s+1$, and with it the state of every expert from chunk $s+2$ on; Appendix I.2 measures both paths. In language modeling, the structural maxima are $(c-1)\,\ell$ write cells $\{(s,r) : s < c,\ r \le \ell\}$ for a layer-private read and $(c-1)\,L$ cells $\{(s,r) : s < c\}$ for a universal read; realized reach in routed pools also requires active routes. The difference, $r > \ell$, identifies the potential top-down routes. □

I.2 Validation: probe, training control and interventions

Proposition 1 states which dependencies can exist; Figure 4 tests them in trained models with a read-output gradient probe (panel a), a training control that cuts gradients through other-layer write contributions while keeping the forward pass (panel b), and inference-time interventions that remove

writes from fixed checkpoints (panels b and c). The language-model diagnostics use the first 32K tokens of single documents from the 3,000-book Books3 cohort of Appendix F, without packing.

Read-output gradient probe. We add independent zero-valued perturbations at the write inputs (k, v, η) of every (chunk, layer) cell, without detaching the graph that produces them. The probe is a fixed random projection of layer ℓ's fast-weight read in the last chunk, taken before the TTT scale and the output projection and summed over the last 32 positions. Its coefficients are independent Rademacher signs (± 1 with equal probability), sampled once and shared across documents and models. For each cell we sum the squared gradient of the probe over the cell's write inputs and report it as a share of the total over all cells. The box in Figure 4a contains the cells of the layers above ℓ in earlier chunks, the top-down routes into the read; a boxed cell counts as reached when its share is nonzero in every document. Figure 4a applies the $\ell = 1$ probe to the 124M TTT-MoE and uTTT-MoE (12 layers, 8 chunks of 4,096 tokens). It uses 300 documents: 32 evenly spaced ones plus 268 drawn without replacement from the remaining 2,968.

Probe checks. Randomly initialized six-layer probe models serve as checks. Finite-difference perturbations confirm the tested zero cells of the layer-private models, and routing controls in which no two layers share an expert isolate the additional paths through pooled momentum.

Cross-layer gradient cut. The training intervention gives each reading layer ℓ its own backward stream of the pool, including master weights and momentum buffers. All streams share the learned initialization. Writer r computes $G_{s,r}$ at its own stream's chunk-start state $W_{\langle r\rangle}^{(s-1)}$; its predicted momentum coefficient is $\mu_{s,r}$ from (7). Stream ℓ then uses

$$\begin{aligned} \mu_{\langle \ell\rangle}^{(s)} &= \frac{1}{L}\Big(\mu_{s,\ell} + \textstyle\sum_{r\neq\ell} \mathrm{sg}(\mu_{s,r})\Big), \\ W_{\langle \ell\rangle}^{(s)} &= \mathrm{Update}\left(W_{\langle \ell\rangle}^{(s-1)},\ G_{s,\ell} + \textstyle\sum_{r\neq\ell} \mathrm{sg}(G_{s,r});\ \mu_{\langle \ell\rangle}^{(s)}\right). \end{aligned} \tag{19}$$

Here sg is the stop-gradient, and the argument after the semicolon makes the momentum coefficient of the stateful update explicit. Layer ℓ reads $W_{\langle \ell\rangle}^{(c-1)}$ in chunk c. In the forward pass sg is the identity, and evaluation preserves the original writer order and reductions, so every stream equals the full shared state and the logits are unchanged. In backward, the update retains the gradient to each stream's previous state and routes current contribution gradients only to its own $G_{s,\ell}$ and to its own coefficient with the mean's factor $1/L$. Stopping $G_{s,r}$ for $r \neq \ell$ removes both its write-input derivative and its state derivative $\partial G_{s,r}/\partial W$, changing the transition Jacobians. Same-layer and residual-stream paths remain available, but gradients to earlier same-layer writes can change magnitude. We call (19) the *cross-layer gradient cut*. Stopping G alone would leave the additional producer paths through input-dependent pooled momentum; the implemented cut also stops the other layers' coefficient contributions. On the probe models the tested cut preserves the logits exactly and removes the gradient from every boxed cell while keeping the reading layer's own row.

Training controls. We train the 124M uTTT-MoE configuration of Table 1 with the cross-layer gradient cut and call the result Cut. It keeps the configuration, batch shape (one node of eight GPUs, four sequences per GPU), 10,240 steps and evaluation stream; only the backward pass differs.

Inference route knock-outs. A route is one (writer, reader) pair of layers. To remove routes, each reading layer receives its own copy of the pool. At each write-enabled boundary, writer r computes $G_{s,r}$ once at the chunk-start state of the copy it reads; each reader's copy then receives the sum of these contributions over the writers permitted by $\mathrm{allowed}[\text{reader}, \text{writer}]$. In language modeling, its momentum coefficient is recomputed as the mean over those same allowed writers; NVS omits momentum (Appendix C.2). Thus reading only own writes uses the reader's own contribution and, in language modeling, its own coefficient. In the language-model no-write condition, every copy has no allowed writers, skips the entire update including its momentum buffers, and remains at the learned initialization. Allowing every route reproduces the unmodified model exactly. The trained parameters stay fixed while the permitted writes and, in language modeling, their pooled coefficient determine each copy's forward updates.

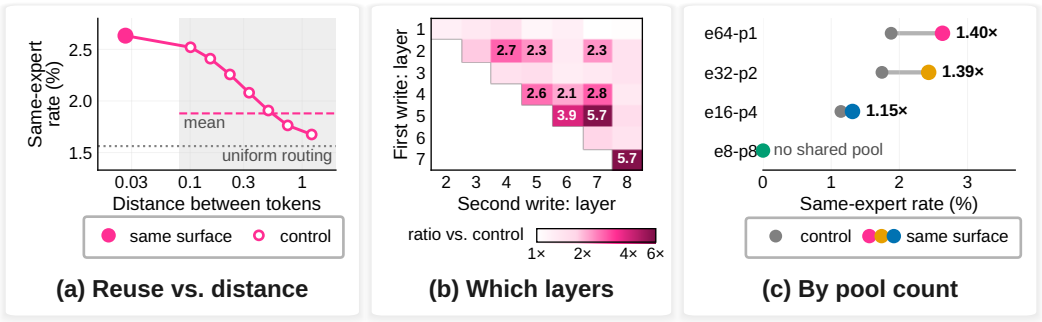


Figure I1: **Quantitative reuse of shared experts by redundant tokens** (GSO, 1,019 objects, 8.6M token pairs). All panels compare the write routing of an earlier-view token at one layer with that of a later-view token at a deeper layer, over 28 layer pairs and 8 heads. *(a)* Same-expert rate versus 3D Euclidean distance for e64-p1; objects are scaled to a longest bounding-box side of 1.8. Filled: geometry-matched pairs; open: controls on the object surface from the same later view, at distance ≥ 0.08, grouped by distance. Dashed: pooled control rate; dotted: uniform routing ($1/64$). *(b)* Same-surface/control ratio for each layer pair in e64-p1, maximum over the 8 heads. Colors are logarithmic; only ratios ≥ 2 are labeled. *(c)* Same-surface and control rates for four sharing partitions, with their ratios. Rates use all 28 layer pairs, including pairs in different pools that cannot share a physical expert.

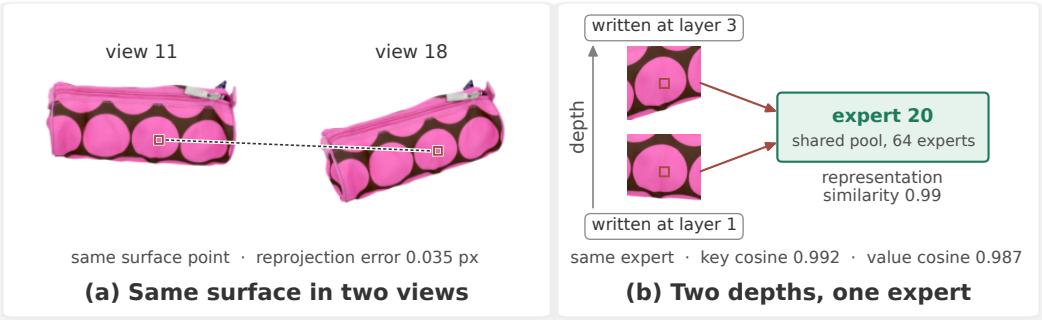


Figure I2: **A redundant token pair reuses one physical expert across depths** (uTTT-MoE-e64-p1, one GSO object). *(a)* Two source views of the object; the red squares mark two tokens that see the same surface point (mean reprojection error 0.035 source pixels). *(b)* The same two tokens, placed at the depth at which each is written into the pool: the view-11 token at layer 1 and the view-18 token at layer 3. Red arrows mark their writes; both are routed to expert 20 of the 64-expert shared pool. Their keys and values have cosine similarities of 0.992 and 0.987, a mean representation similarity of 0.99.

Language modeling. Figure 4b applies two conditions to the Full and Cut checkpoints: freezing state writes, which disables every boundary update of the pool, and reading only own writes, which removes every cross-layer route. We use the first 1,024 documents of the cohort. The loss increase of a condition is measured against the same checkpoint's unmodified predictions: we average the loss within each 4K chunk, then over chunks 2–8, then over documents. Chunk 1 reads only the initial state and is identical in every condition. The unmodified tail PTL of Full, Cut and TTT is averaged over positions 30,000–32,766 of the same documents.

View synthesis. Figure 4c removes the 64 writer–reader routes of uTTT-MoE-e64-p1 one at a time, including the eight same-layer routes on the diagonal, and reports the mean paired PSNR drop, unmodified minus knock-out, over the 1,019 GSO objects and their 23 target views.

I.3 BEHAVIOR: EXPERT REUSE BY REDUNDANT TOKENS

We now turn from dependence to behavior: are repeated observations of the same surface routed to the same shared experts across views and depths? Two source views that see the same surface region produce geometry-matched tokens; we call such a pair *redundant*. We report population statistics over all GSO objects, then one example pair. Over the $1{,}019$ objects, 8.6M same-surface pairs are compared at 224 (layer pair, head) combinations each: the earlier view's token, written at some layer, against the later view's token written at a deeper layer. In uTTT-MoE-e64-p1 the two are routed to the same physical expert in 2.63% of the comparisons, against 1.88% for a control token from the same later view at least one token spacing away from the first token's surface point, and 1.56% for uniform routing over 64 experts. The same-surface/control ratio is 1.40 [1.39, 1.41] (95% interval from $10{,}000$ object resamples) and exceeds 1 on every object.

Figure I1a shows that the same-expert rate decreases with the 3D distance between the two tokens, and panel b shows which layer pairs have strong per-head enrichment. Across sharing partitions (panel c), the same-surface/control ratio is $1.40\times$, $1.39\times$ and $1.15\times$ for e64-p1, e32-p2 and e16-p4, respectively; the layer-private e8-p8 shares no physical expert across layers. The absolute rates include all 28 layer pairs, so their decrease also reflects that fewer pairs share a pool. The later token's read routing, taken while its view is rendered from camera rays alone and before its image is written, also favors the earlier token's write experts, with a same-surface/control ratio of 1.34 in e64-p1. These statistics measure how often repeated observations are routed together; Figure 4c separately tests how prediction depends on individual state-read routes. Figure I2 shows one such pair.