Universal Test-Time Training

Anonymous authors
Paper under double-blind review

Abstract

Test-Time Training (TTT) compresses context into fast weights that are updated online and queried as recurrent memory. In existing designs these fast weights are *private*: every layer owns a state no other layer touches. We argue this is an arbitrary restriction and study its alternative, *universal fast weights*, in which the whole depth stack shares one state, or one pool of states, rather than L private ones. Sharing lets a fixed fast-weight budget be spent as one wide operator that every layer applies — more memory FLOPs per token — or as one pool that every layer routes into, giving a larger effective state, allocated across depth by routing rather than fixed at design time, updated as one grouped kernel rather than L small ones. We instantiate this as *uTTT-Dense*, a shared dense operator; *TTT-MoE*, a routed pool that stays layer-local; and *uTTT-MoE*, a globally shared pool. We further observe that a universal TTT model is secretly a fully recurrent model: one state persisting across depth makes the layer index a second recurrence axis, so the L-layer stack unrolls into a single recurrence over (chunk, layer) pairs and state written deep at one chunk is read shallow at the next; layer-private TTT is the fixed-partition special case. Across large-chunk language modeling and LVSM-style novel view synthesis, moving fast weights from layer-private, to layer-local routed, to a globally shared pool improves long-context retrieval monotonically — RULER accuracy from 8.5 to 15.5 at 124M and 23.2 to 27.8 at 760M, and object-level rendering from 24.6 to 25.5 dB — though the advantage concentrates at shorter contexts and does not yet hold at the longest evaluated length.

1 Introduction

Long-context modeling trades explicit memory for compressed state. Attention keeps the full key-value history at growing cost (Vaswani et al., 2017); recurrent networks, linear attention, and state-space models compress it into a bounded state (Hochreiter & Schmidhuber, 1997; Katharopoulos et al., 2020; Gu & Dao, 2023; Dao & Gu, 2024; Yang et al., 2024). Test-Time Training (TTT) compresses one step further, storing context in *fast weights* that are optimized online during the forward pass and queried as memory (Hinton & Plaut, 1987; Schmidhuber, 1987; Schlag et al., 2021; Sun et al., 2024; Behrouz et al., 2024); large-chunk TTT amortizes those updates so that nonlinear fast-weight state is practical at long context (Zhang et al., 2025).

Once fast weights are the memory, a design question appears that is orthogonal to how they are updated: *who owns them*. Existing TTT architectures answer it the same way. Each layer instantiates its own fast-weight tensor, updates it from its own keys and values, and reads it with its own queries; no other layer touches that state (Sun et al., 2024; Zhang et al., 2025; Tandon et al., 2025). The recurrent memory of an L-layer model is therefore not one memory but L disjoint ones, evolving in parallel and communicating only through the token stream. This is inherited from the transformer parameter layout, not from any property of test-time optimization, and it has three consequences: a fixed fast-weight budget is divided by L, so each state is small and the operator every token applies is narrow; the split of memory across depth is fixed at design time, so a layer cannot draw on a neighbor's state; and the implementation manipulates L small tensors, so the fast-weight path runs as many small per-layer operations.

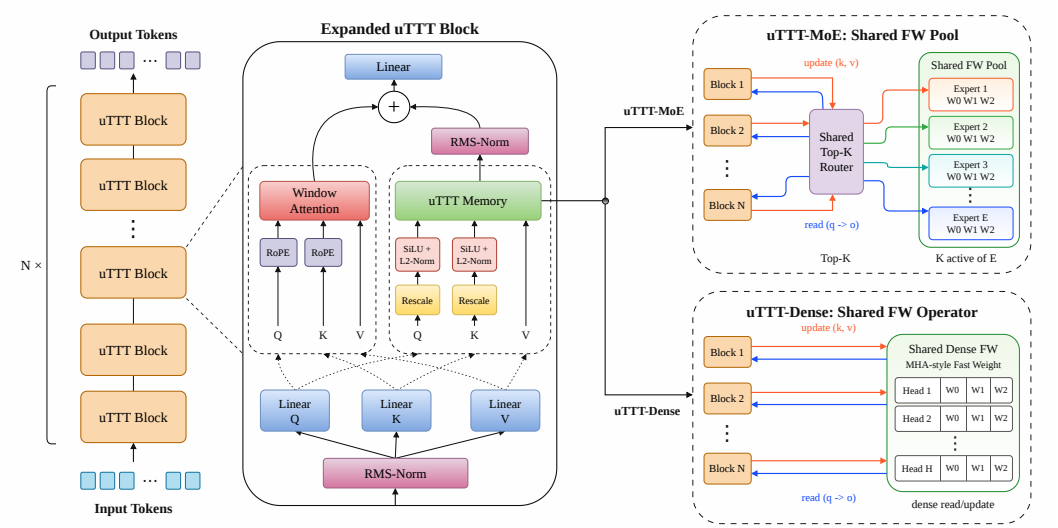

Figure 1: **Universal Test-Time Training.** *Left:* the model is a stack of N uTTT blocks. *Middle:* a block keeps its own normalization, projections, and local window attention private, and sends its projected queries, keys, and values to the uTTT memory path. *Right:* the memory itself is shared across the whole stack. uTTT-MoE (top) exposes it as a pool of fast-weight experts that every block addresses through a shared top-K router, so K of E experts are active per token head; uTTT-Dense (bottom) exposes it as a single MHA-style dense fast-weight operator that every block reads and updates in full. Blue arrows are reads ($q \to o$, Eq. (5)); orange arrows are updates from keys and values (Eq. (6)). Because the same memory is written by every depth, the stack forms one recurrence over (chunk, layer) rather than L independent ones (Section 4.6).

This paper studies the alternative. We call fast weights *universal* when the whole depth stack reads and writes one shared state, or one shared pool of states, rather than L private ones. Universality helps for a different reason in each regime a TTT model is built in.

Dense TTT: universal sharing buys fast-weight FLOPs. In a dense TTT layer every token applies the entire state, so state size and per-token fast-weight compute are one quantity. Splitting a fixed budget L ways gives each layer a narrow operator; sharing lets the same budget be assembled into one wide operator that all L layers apply, multiplying the fast-weight FLOPs spent on memory without adding a parameter. Universal dense sharing thus scales the compute of the memory path — the axis dense TTT cannot otherwise scale independently of parameter count (Section 4.2).

Sparse TTT: universal sharing buys effective state and kernel structure. A routed pool of E experts already separates capacity from active compute: it stores E times an expert's state while each token head activates only $K \ll E$. A layer-local pool, though, still lets a forward reach only its own layer's share; a universal pool lifts this ceiling, so the effective state visible at each forward grows by up to $L\times$ at equal parameters, and how much each depth claims becomes a learned allocation rather than a fixed partition. Sharing also reshapes the computation: because all layers write one bank, their per-chunk updates are gradients on the same tensors and can be applied as a single grouped operation instead of L small ones (Sections 4.4–4.5).

Relation to prior sharing. Sharing parameters across depth is established for slow weights — Universal Transformers, ALBERT, and their MoE variants (Dehghani et al., 2019; Lan et al., 2020; Tan et al., 2023; Csordás et al., 2024) — and recently for the expert pool of a transformer MoE (Huang et al., 2026; Chen et al., 2026). We take this to fast weights, where

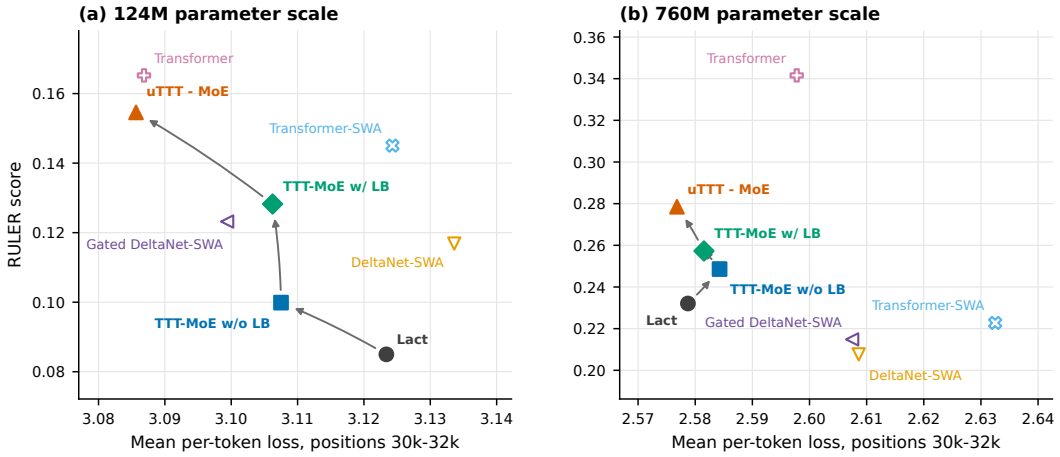

Figure 2: **Per-token loss versus long-context retrieval**, at (a) 124M and (b) 760M parameters. Each point is one model; the horizontal axis is per-token loss averaged over positions beyond 30K in 32K-token sequences and the vertical axis is RULER accuracy averaged over 4K–32K, so the upper-left corner is best. Making the fast-weight pool universal (uTTT - MoE) moves the model toward that corner relative to its balanced and unbalanced layer-local counterparts (TTT-MoE w/ LB and TTT-MoE w/o LB) and to the layer-private dense baseline (Lact), at both scales. Transformer, with full attention, is the one model reaching higher retrieval, at the cost of a key-value cache that grows with context.

the shared object is not a static parameter store but the model's recurrent state, so sharing it changes what the model can remember; Section 2 gives the full comparison.

What we build. We instantiate universal fast weights in three architectures. *uTTT-Dense* shares a single dense operator across all layers. *TTT-MoE* turns a layer's fast weights into a routed expert pool but keeps it private to the layer, and *uTTT-MoE* promotes that pool to one global bank every layer-head routes into; TTT-MoE is the controlled midpoint, so the step to uTTT-MoE isolates universality from routing. We train with an *aggregated* update schedule that reads a common per-chunk snapshot and applies one grouped write over the bank, which is what lets the shared-memory path run as a single large kernel; a *sequential* schedule that writes after each layer is also possible and left as an ablation (Section 4.5).

Universal TTT is secretly a fully recurrent model. Sharing state across depth changes the model class, not just the parameter layout. When one state persists across layers, depth becomes a second recurrence axis: the state written at layer ℓ of chunk c is read at layer $\ell + 1$ of the same chunk and, at the chunk boundary, at layer 1 of chunk $c + 1$. Unrolling depth and time together, the L-layer stack collapses into a *single* recurrence over (c, ℓ) pairs whose carried state is the shared bank. The memory then receives L optimization steps per chunk rather than one, and state written deep in the stack can be read shallow at the next chunk — a top-down path no layer-private design has. Layer-private TTT is the special case in which the bank is partitioned into L disjoint blocks with fixed routing (Section 4.6).

Instantiations. We evaluate the framework where fast-weight memory is stressed in two different ways. The primary setting is large-chunk autoregressive language modeling (Zhang et al., 2025), where the token stream is split into large causal chunks and ownership is measured through long-context retrieval and per-token loss. We then repeat the question in LVSM-style novel view synthesis (Jin et al., 2024; Sajjadi et al., 2022; Wang et al., 2025b), where posed images are written into the fast weights as scene memory and pose-only cameras query that state, so the chunk structure is explicit and the memory demand is spatial rather than lexical.

Contributions. (i) We identify fast-weight ownership as a design axis of TTT and argue that the universal setting, not the layer-private one, is the natural default. (ii) We give three architectures — uTTT-Dense, TTT-MoE, and uTTT-MoE — and explain why universality helps in the dense regime (fast-weight FLOPs) and in the sparse regime (effective state, learned depth allocation, and grouped-kernel updates). (iii) We show that a universal TTT model is a fully recurrent model over the joint (chunk, layer) index, with layer-private TTT as a fixed-routing special case. (iv) We instantiate the framework in language modeling and novel view synthesis, with controlled comparisons that isolate the sharing domain from routing and balancing.

2 Related Work

Test-time training, fast weights, and recurrent models. Fast weights are a classical mechanism for rapidly writable memory (Hinton & Plaut, 1987; Schmidhuber, 1987); modern Test-Time Training realizes them as neural parameters optimized online and queried as recurrent state (Schlag et al., 2021; Sun et al., 2024; Behrouz et al., 2024; von Oswald et al., 2025; Wang et al., 2025a; Tandon et al., 2025). TTT therefore belongs to the family of models that compress an unbounded history into a bounded carried state — recurrent networks, linear attention, and state-space models (Hochreiter & Schmidhuber, 1997; Katharopoulos et al., 2020; Gu & Dao, 2023; Dao & Gu, 2024; Yang et al., 2024) — distinguished by a state that is a parameter vector and a transition that is a gradient step rather than a fixed linear map. Large Chunk TTT (LaCT) amortizes the update over large chunks, raising arithmetic intensity and making nonlinear fast-weight state practical at long context (Zhang et al., 2025). All of these, LaCT included, keep the state layer-private, so the recurrence is block-diagonal across depth. We leave the update rule and chunk granularity untouched and change only ownership, which turns those L block-diagonal recurrences into one recurrence that also runs along depth.

Mixture-of-experts. Sparse MoE layers decouple total capacity from active per-token computation by routing each token to a few experts (Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2022; Clark et al., 2022; Jiang et al., 2024). The same principle has been applied to memory: MoM routes among several linear memory states (Du et al., 2025), and LaCT-style routing gives large-chunk TTT fast-weight experts whose address space is nonetheless confined to one layer (Zhang et al., 2025). We adopt sparse routing for the same reason — it separates stored from active state — but treat the routed experts as the recurrent memory of the whole model, which additionally forces a choice of how reads and updates from different depths interleave on shared experts.

Universal models and cross-layer sharing. Universal Transformers tie a block's parameters across all layers and read depth as recurrence with adaptive halting (Dehghani et al., 2019); ALBERT ties parameters for efficiency (Lan et al., 2020). This work establishes depth-wise weight sharing as a viable inductive bias, and a shared block applied repeatedly as a recurrent computation. Our claim is analogous in form but concerns a different object: a Universal Transformer shares slow weights that are fixed during a forward pass, whereas uTTT shares a state rewritten several times within one forward pass, so sharing it couples the layers dynamically rather than only tying their parameters. We do not require slow weights to be tied.

Universal mixture-of-experts. Closest to us is recent work that makes a transformer MoE's expert pool universal across depth. UniPool replaces layer-private expert sets with one globally shared pool addressed by independent per-layer routers, balanced at the pool level (Huang et al., 2026); MoUE shares a pool of universal experts so depth becomes reusable virtual width under a fixed activation budget (Chen et al., 2026). Partial variants relax rather than remove the sharing domain: ReXMoE reuses experts across adjacent layers (Tan et al., 2025), tied-expert models share the expert sub-block across groups of layers (Jaggi, 2026), path-constrained MoE shares routers instead of experts (Gu et al., 2026), and MoEUT and Sparse Universal Transformers pair sparse experts with a layer-tied backbone (Csordás et al., 2024; Tan et al., 2023). These directly motivate uTTT-MoE;

the difference is what the pool holds. In a transformer MoE the shared experts are static functions, so universality is parameter reuse; in uTTT-MoE the shared experts are the recurrent state, so universality also fixes which layer reads what another layer wrote, and at which chunk.

Novel view synthesis as a memory testbed. LVSM-style novel view synthesis compresses posed observations into a learned scene state and renders target cameras from it (Jin et al., 2024; Sajjadi et al., 2022; Wang et al., 2025b). This makes it a clean stress test for fast-weight memory: update chunks must accumulate cross-view information in a bounded state, and query chunks must retrieve what a new pose needs. We use Objaverse/GSO and DL3DV (Deitke et al., 2023; Downs et al., 2022; Ling et al., 2024) not for a leaderboard but because the task's read/write structure is explicit enough to make ownership measurable.

3 Preliminaries

This section fixes the notation for TTT memory, for chunked recurrence, and for the state and compute budgets that the rest of the paper holds constant when comparing ownership schemes.

3.1 Test-Time Training

Let $x_{1:T}$ be a token sequence processed by a stack of L blocks. Block ℓ uses slow weights θ_ℓ, learned by ordinary training, to form projected features $(q_{\ell,i}, k_{\ell,i}, v_{\ell,i})$ for token i. Attention would store the context as the accumulated keys and values; a TTT layer instead stores it in *fast weights* W, a set of neural parameters that is optimized during the forward pass, and reads it through a parametric readout $f_W(\cdot)$.

Read and update. Reading applies the current fast weights to a query,

$$o_i = f_W(q_i), \tag{1}$$

and writing performs one step of an online optimizer on a self-supervised objective that fits keys to values,

$$W \leftarrow \text{Update}\Big(W,\ \nabla_W \textstyle\sum_i \eta_i\, \mathcal{L}\big(f_W(k_i), v_i\big)\Big), \tag{2}$$

where η_i is a learned or fixed per-token weight and Update may include momentum, weight decay, or a normalized step. The recurrent state of the model is thus a set of parameters, and the transition is a descent step rather than a fixed linear map. Whether f_W is linear or a small MLP, and whether Update is plain gradient descent or a preconditioned rule, is orthogonal to the question we study.

Ownership. Equations (1)–(2) say nothing about which W a given block uses. Standard practice instantiates one fast-weight tensor W_ℓ per block, so block ℓ reads and writes only W_ℓ. We call this the *layer-private* convention, and it is the assumption this paper removes.

3.2 Chunk Recurrence

Updating fast weights once per token is expensive and has poor arithmetic intensity, because each token triggers a small optimizer step on the whole state. Large-chunk TTT instead partitions the sequence into chunks $C_1, \ldots, C_N$ and updates once per chunk (Zhang et al., 2025): all tokens in a chunk read the same state, their update contributions are accumulated, and one write occurs at the chunk boundary. Writing $W^{(c)}$ for the state after chunk c, a layer-private model evolves as

$$o_{\ell,i} = f_{W_\ell^{(c-1)}}(q_{\ell,i}), \quad i \in C_c, \qquad W_\ell^{(c)} = \text{Update}\Big(W_\ell^{(c-1)},\ \nabla \textstyle\sum_{i\in C_c} \eta_{\ell,i}\, \mathcal{L}\big(f(k_{\ell,i}), v_{\ell,i}\big)\Big). \tag{3}$$

Two properties of (3) matter later. First, the recurrence is *block-diagonal in depth*: W_ℓ depends only on the history of W_ℓ, and layers exchange information only through the token activations within a chunk. Second, the state advances exactly N times over a sequence of N

chunks, regardless of how deep the model is. What counts as a chunk is setting-specific — a posed image or a block of tokens — and is defined for each in Sections 4.7 and 4.8.

3.3 Total State, Active State, and Update Steps

Two quantities separate once fast weights are routed. *Total state* is all fast-weight memory maintained for a sequence; *active state* is the subset a single token head reads or updates. Dense TTT ties them — every token applies the whole state, so widening the state widens the compute by the same factor — while sparse fast weights separate them: with E experts of size S and $K \ll E$ active, total state scales as ES and active compute as KS. A third quantity is implicit in (3) and becomes free only once fast weights are shared across depth: the number of update steps the memory receives per chunk, one for layer-private models and up to L for universal ones.

4 Method

Universal Test-Time Training (uTTT) removes the layer-private convention of Section 3. As shown in Figure 1, every block keeps its own slow weights, projections, and local attention, but the fast weights it reads and writes are drawn from one memory shared by the whole depth stack. Section 4.1 defines the shared memory and its routing, Sections 4.2–4.4 give the three architectures, Section 4.5 the update schedules, and Section 4.6 the recurrent-model view.

4.1 Universal Fast Weights

Let $\mathcal{B} = \{W_e\}_{e=1}^{E}$ be a bank of fast-weight modules, each with readout $f_{W_e}(\cdot)$. The bank is a property of the *model*, not of any block: it is allocated once and persists across depth and across chunks. A block ℓ and head h do not own state; they own an *address*, a routing function

$$\mathcal{R}_{\ell,h}(z) \subseteq \Omega_\ell \subseteq \{1, \dots, E\}, \tag{4}$$

into the *visible domain* Ω_ℓ, the subset of the bank layer ℓ may address. Token i at (ℓ, h) reads a gated combination of the addressed modules,

$$o_{\ell,h,i} \;=\; \sum_{e \,\in\, \mathcal{R}_{\ell,h}(q_{\ell,h,i})} g_{\ell,h,i,e}\; f_{W_e}(q_{\ell,h,i}), \tag{5}$$

and, at the end of chunk C, the addressed modules are updated by the accumulated contributions of every layer-head that routed to them,

$$W_e \leftarrow \mathrm{Update}\Big(W_e,\; \nabla_{W_e} \sum_{\ell,h} \sum_{\substack{i \in C \\ e \,\in\, \mathcal{R}_{\ell,h}(k_{\ell,h,i})}} g_{\ell,h,i,e}\, \eta_{\ell,h,i}\, \mathcal{L}\big(f_{W_e}(k_{\ell,h,i}), v_{\ell,h,i}\big)\Big). \tag{6}$$

Read routing and update routing share the address space but use separate projections, so a layer-head may read from one part of the bank and write to another. The design choices are the bank size, each layer's visible domain Ω_ℓ, how many modules a head activates, and when (6) is applied relative to (5) across depth.

Ownership as a routing constraint. The visible domain makes prior designs special cases. Layer-private dense TTT is $E = L$, $\Omega_\ell = \{\ell\}$, and $\mathcal{R}_{\ell,h} \equiv \Omega_\ell$: the routing is constant and the domains partition the bank. Layer-private routed TTT keeps disjoint Ω_ℓ but lets $\mathcal{R}_{\ell,h}$ vary with the input inside its block. Universal designs set $\Omega_\ell = \{1, \dots, E\}$ for all ℓ. What matters is therefore not whether fast weights are routed but how large the visible domain is relative to the bank, and we sweep it from fully partitioned to fully shared.

4.2 uTTT-Dense

uTTT-Dense is the case $E = 1$: a single dense operator W, visible to every layer, with no routing to decide. Each block projects its own (q, k, v), reads $f_W(q)$, and contributes to the

update of W. Suppose the layer-private baseline gives every block a state of size S, so the model stores LS fast-weight parameters and a token performs $\Theta(S)$ fast-weight work at each layer. Two matched variants follow. The **FLOPs-matched** variant (W of size S) makes the shared operator one baseline layer's size: total parameters drop by L while per-token compute is unchanged, isolating sharing from scale. The **size-matched** variant (W of size LS) assembles the whole depth-wide budget into one wide operator: total parameters equal the baseline, but every token now applies an L-times-wider operator, so per-token fast-weight compute rises by L. The size-matched variant is the one the dense argument predicts should help — at fixed parameter count it converts memory fragmented across depth into one large state and scales the compute of the memory path. Its cost is that dense reads and writes are unconditional, so state and compute stay coupled, which motivates the sparse variants.

4.3 TTT-MoE: Routed but Layer-Local Fast Weights

TTT-MoE replaces block ℓ's single state with a private pool Ω_ℓ of E_{loc} experts, disjoint across blocks, and routes each token head to K of them via (4), with reads and updates restricted to a single ℓ. This decouples state from compute within a layer — the layer stores E_{loc} experts' worth of state but a token pays for K — and adds specialization, since different experts can capture different aspects of the context. It does not change ownership: the visible domain is still a depth-indexed partition, the reachable state at any forward is $E_{\text{loc}}S$ rather than the model's $LE_{\text{loc}}S$, and the recurrence stays block-diagonal. TTT-MoE is therefore the controlled midpoint of the design space; it isolates the benefit of *routing*, so the remaining difference to uTTT-MoE isolates the benefit of *universality*.

4.4 uTTT-MoE: A Universal Fast-Weight Expert Pool

uTTT-MoE promotes the pool to one bank of E experts every layer-head may address. Each block forms routing features with layer-specific projections ϕ_ℓ and per-head normalization, and selects experts by inner product against learned expert addresses r_e:

$$\mathcal{R}_{\ell,h}(z) \;=\; \text{TopK}\left(\left\{\phi_\ell(z_{\ell,h})^\top r_{\rho(\ell),e}\right\}_{e\,\in\,\Omega_{\rho(\ell)}},\; K\right), \tag{7}$$

where a softmax over the selected logits produces the gates $g_{\ell,h,i,e}$ of (5). Read routing projects the query stream and update routing the key stream; the per-head normalization matters more here than with layer-local pools, because logits produced at different depths compete on one address space and would otherwise be incomparable in scale. The map ρ assigns each layer to a pool group and thereby sets the sharing topology: *local* ($\rho(\ell) = \ell$) recovers TTT-MoE, *partitioned* groups contiguous layers into p banks (Tan et al., 2025; Jaggi, 2026), and *global* shares one bank across all layers (Huang et al., 2026; Chen et al., 2026).

What universality provides. Three properties follow, the sparse counterpart of the dense FLOPs argument. *(i) Effective total state:* a layer's reachable memory grows from its own share to the entire bank, so at fixed parameters the state a forward can draw on is larger by up to $L\times$. *(ii) Depth-allocated capacity:* because $\mathcal{R}_{\ell,h}$ is learned and input-dependent, the fraction of the bank each depth claims is set by the router, not the architecture; a layer that needs more memory routes to more experts, and layers with similar needs share experts instead of keeping redundant copies. *(iii) Cross-depth reuse:* an expert updated at layer ℓ of chunk c may be addressed at layer $\ell' < \ell$ of chunk $c+1$, so memory written high in the stack becomes available low in the stack at the next chunk.

Load balancing. A shared bank makes balancing a model-level concern, since one depth can monopolize an expert, and we enforce it at the pool level. The auxiliary-loss variant penalizes the deviation from uniform of expert-utilization statistics pooled over all layer-heads. The auxiliary-loss-free variant instead maintains a per-expert additive bias, applied to the routing logits before the top-K selection and adjusted from that expert's running utilization; the bias steers selection but leaves the gate values of (5) unchanged. Both match the pool-level balancing of globally shared transformer expert pools (Huang et al., 2026; Chen et al., 2026).

4.5 Update Schedules and Grouped Execution

With a shared bank the order of reads and writes across depth becomes a real choice. Writing $W^{(c,\ell)}$ for the bank after layer ℓ of chunk c, the *sequential* schedule writes after each layer,

$$o_{\ell,i} = f_{W^{(c,\ell-1)}}(q_{\ell,i}), \qquad W^{(c,\ell)} = \text{Update}\left(W^{(c,\ell-1)}, G_{c,\ell}\right), \tag{8}$$

with $G_{c,\ell}$ the update contribution of layer ℓ over chunk C_c and $W^{(c,0)} = W^{(c-1,L)}$: the state advances L times per chunk, and a layer sees everything written below it in the same chunk. The *aggregated* schedule has all layers read the chunk-start snapshot and sums their contributions into one write,

$$o_{\ell,i} = f_{W^{(c-1)}}(q_{\ell,i}) \;\; \forall \ell, \qquad W^{(c)} = \text{Update}\left(W^{(c-1)}, \textstyle\sum_{\ell=1}^{L} G_{c,\ell}\right), \tag{9}$$

so the state advances once per chunk, as in the layer-private case, but integrates evidence from every depth.

Grouped execution. The aggregated schedule is what makes the fast-weight path cheap. A layer-private model performs L separate optimizer steps per chunk, each on a small state, and in the routed case L separate dispatch/gather sequences — small, launch-bound operations that underutilize the accelerator. Under (9) the per-layer contributions are gradients on the same tensors, so they accumulate and apply as one large update over the bank at the chunk boundary, and for uTTT-MoE the routed writes form one grouped scatter rather than L small ones. The read path stays sequential in depth, since layer ℓ's queries depend on layer $\ell-1$'s output, but all reads hit one snapshot that stays resident across the traversal. The sequential schedule (8) gives up this grouping for stronger within-chunk recurrence; we train with the aggregated schedule and treat the choice as an ablation.

4.6 A Universal TTT Model Is Secretly a Fully Recurrent Model

Sharing state across depth changes the model class, not just the parameter layout. Indexing the (c,ℓ) pairs in execution order by $t = (c-1)L + \ell$, the sequential schedule (8) makes the shared bank obey

$$W^{(t)} \;=\; \text{Update}\left(W^{(t-1)}, G_t\right), \qquad o^{(t)} \;=\; f_{W^{(t-1)}}\left(q^{(t)}\right), \tag{10}$$

a *single* recurrence of length NL whose carried state is the entire fast-weight memory. The cell at step t uses slow weights $\theta_{\ell(t)}$ with $\ell(t) = 1 + ((t-1) \bmod L)$, a period-$L$ modulation; tying $\theta_1 = \cdots = \theta_L$ removes it and gives a homogeneous, Universal-Transformer-style recurrence whose state is its fast weights (Dehghani et al., 2019), which we do not require. Three consequences follow.

- **Depth is extra test-time optimization.** The layer-private recurrence (3) advances each state N times over a sequence; (10) advances one state NL times. Since each step descends the TTT objective, depth supplies more inner-loop optimization on one memory rather than initializing L separate optimizations.
- **Information can flow backward in depth.** In (3) information reaches layer ℓ' from layer $\ell > \ell'$ only through the loss during training; at inference the path is strictly bottom-up. Under sharing, layer ℓ of chunk c writes state that layer $\ell' < \ell$ reads at chunk $c+1$, a top-down pathway with a one-chunk delay.
- **Layer-private TTT is a strict special case.** Setting $E = L$, $\Omega_\ell = \{\ell\}$, and $\mathcal{R}_{\ell,h} \equiv \Omega_\ell$ makes each G_t act on block $\ell(t)$ alone, so the transition of (10) is block-diagonal and factorizes into L non-interacting length-N recurrences that reproduce (3) exactly. The aggregated schedule (9) recovers the same baseline under the same constraint.

Every architecture here thus lies on one axis: how much of the bank each depth may see, from a disjoint partition (layer-private) through contiguous groups (partitioned) to the whole bank (universal).

4.7 Instantiation: Autoregressive Language Modeling

For language modeling, the token stream is split into large causal chunks following the large-chunk TTT recurrence (Zhang et al., 2025). A shifted block-wise update/apply order lets tokens in a chunk read only fast weights written by strictly earlier chunks, preserving causality through the memory path, while sliding-window attention supplies local causal context inside the chunk. Causality is unaffected by sharing, since (8) and (9) both order writes at chunk boundaries; what sharing changes is only which layer may read a given write.

4.8 Instantiation: Novel View Synthesis

In the LVSM-style setting a scene is a sequence of chunks. A posed RGB image, patchified with its ray encoding, forms an update chunk that reads the current bank and contributes to (6); a target camera, encoded by rays alone, forms an apply-only chunk that reads through (5) and renders the held-out view. Each block keeps local window attention over the current chunk, which preserves image structure, while all cross-view information passes through the fast weights. The bank is thus the only cross-chunk scene memory, so ownership is directly observable in rendering quality. Because the number and order of input views vary, the bank must accumulate scene content in an order-robust way. All variants share the same patchification, window size, depth, width, and rendering head; only the fast-weight bank and its routing differ.

5 Experiments: Language Modeling

Long-context language modeling is our primary testbed. It isolates the ownership question in the setting where fast weights are most obviously acting as memory: the chunk structure comes from a causal token stream, sliding-window attention supplies local context, and anything a token needs from far earlier in the sequence must have survived in the recurrent state. It also admits two metrics that stress different things — per-token loss and explicit retrieval — which turn out not to agree.

5.1 Setup

Data and scales. Following the large-chunk TTT language-modeling protocol (Zhang et al., 2025), we train autoregressive models at 32K context with large recurrent chunks on Long-Data-Collections (AI, 2024), at 124M and 760M parameter scales. The smaller scale is used for the wider architectural sweep and the larger scale for controlled pairwise comparisons.

Metrics. We report two complementary quantities. Per-token loss (PTL) is the mean next-token loss over positions at or beyond 30K in 32K-token sequences, evaluated on Book-3 from The Pile (Gao et al., 2020), following the large-chunk TTT protocol (Zhang et al., 2025) and the per-position loss convention of the Forgetting Transformer (Lin et al., 2025); it measures how well the memory path still serves prediction deep into a sequence. RULER retrieval accuracy (Hsieh et al., 2024), averaged over the 4K, 8K, 16K, and 32K settings, measures whether specific content written into the bank can later be recovered. The two are not redundant: a model can lower average loss without improving retrieval.

Variants. The comparison walks a four-point ownership ladder, holding tokenizer, chunk size, sliding-window attention, depth, width, optimizer, and token budget fixed so that only the fast-weight bank differs. LaCT gives every layer one private dense state. TTT-MoE w/o LB replaces that state with a layer-local routed pool. TTT-MoE w/ LB adds auxiliary-loss-free balancing to the same pool. uTTT-MoE promotes the pool to a single globally shared bank with pool-level balancing. Four models outside the fast-weight family bound the comparison: Transformer (full attention) and Transformer-SWA (sliding-window attention), and DeltaNet-SWA and Gated DeltaNet-SWA, which mix a linear-attention state with a

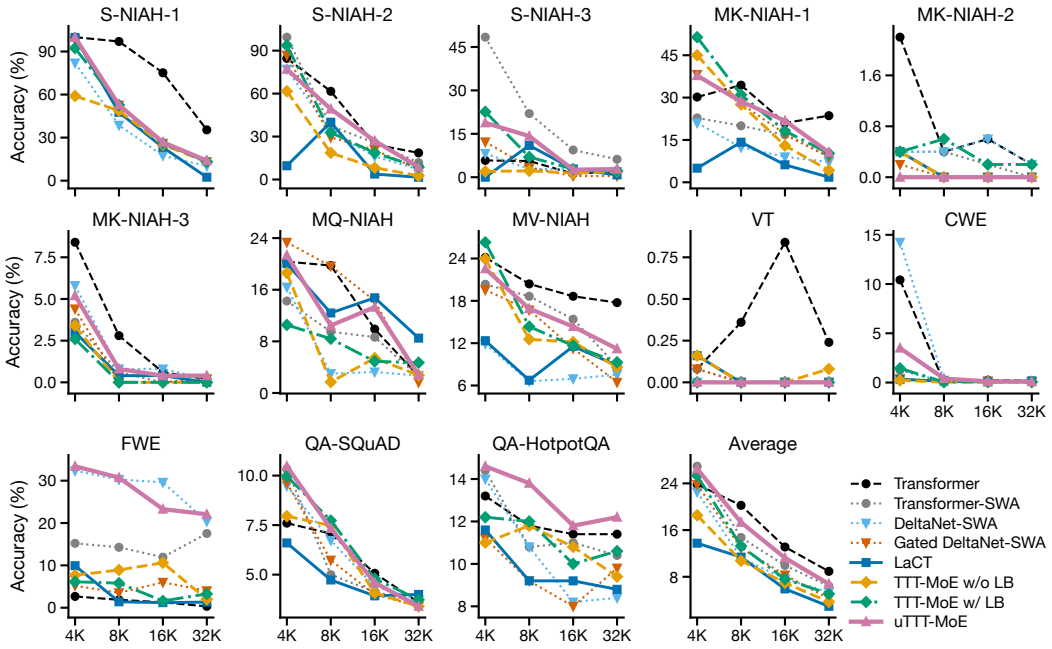

Figure 3: **Per-task RULER accuracy at 124M parameters.** All 13 RULER tasks at 4K–32K; panels 1–13 are the individual tasks, each on its own accuracy range (task difficulty spans more than an order of magnitude), and panel 14 is their unweighted mean, which reproduces the aggregate RULER score of Figure 2. The mean tracks the ownership ladder monotonically, 8.50 → 9.99 → 12.82 → 15.46% from layer-private, to routed, to balanced, to universal; uTTT-MoE is the strongest fast-weight model at every length, with the largest gain over TTT-MoE w/ LB in the middle of the range (+4.1 points at 8K, +3.6 at 16K), and remains below the full-attention Transformer (15.46 vs 16.52%), which keeps a key-value cache that grows with context. The separation is concentrated in single-needle, multi-query, multi-value, and word-extraction tasks; multi-key 2/3 and variable tracking sit near the floor for every method.

sliding window. Only the last step of the ladder changes the sharing domain; the earlier steps vary routing and balancing inside a layer.

Scope. This sweep covers the sparse branch only: uTTT-Dense (Section 4.2) is defined but not evaluated here. Every routed model uses the aggregated schedule, so the sequential schedule of Section 4.5 is not exercised, and we do not report throughput or peak memory, so the cost side of the grouped-execution argument is stated but not measured.

Configurations. Table 1 (Appendix B) lists the architecture and training settings, shared by all variants except the fast-weight bank. The bank is the only difference along the ladder: LaCT keeps one dense fast-weight state per layer; TTT-MoE routes each token head into a layer-local pool of $E = 4$ experts ($K = 1$), either without balancing (w/o LB) or with loss-free balancing (w/ LB); and uTTT-MoE routes into a single global pool ($K = 1$, per-layer router, loss-free balancing) of $E = 48$ experts at 124M and $E = 96$ at 760M. Table 2 gives the full per-model configuration.

5.2 Results

Figure 2 places every model on the two headline axes, Figures 3 and 4 decompose the retrieval score by task and length, and Figure 5 resolves the loss by position. The captions read each figure; here we record only what the three imply jointly. For reference, uTTT-MoE

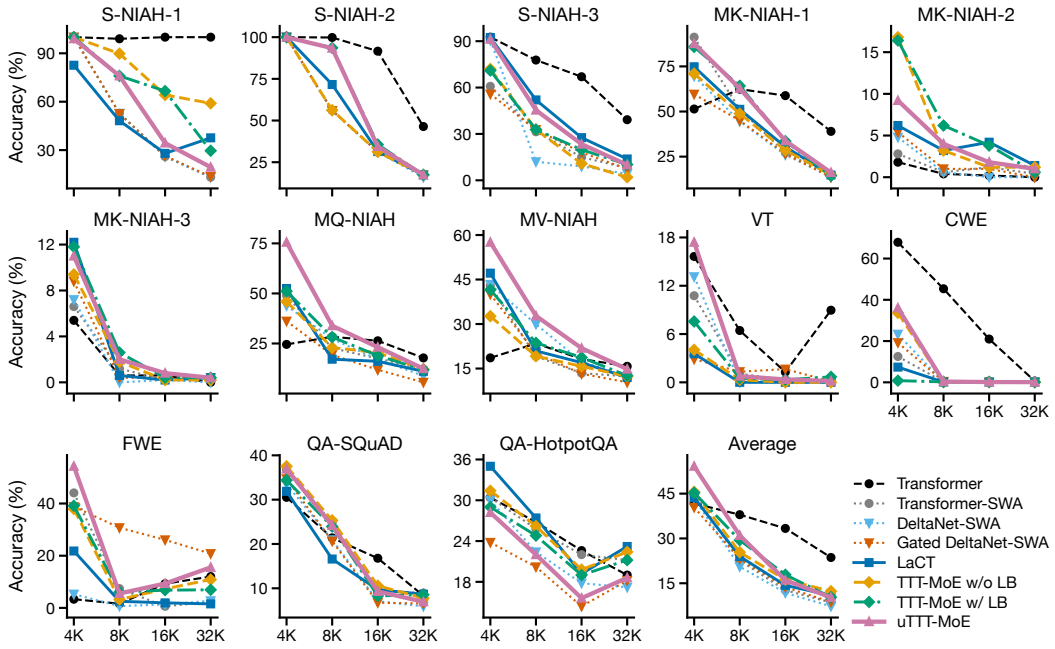

Figure 4: **Per-task RULER accuracy at 760M parameters**, in the layout of Figure 3, at roughly six times the parameter count. The aggregate ordering replicates, 23.21 $\rightarrow$ 24.86 $\rightarrow$ 25.73 $\rightarrow$ 27.85% along the same ladder, so the ownership effect is not an artifact of the small-scale sweep. The per-length decomposition, however, is not uniform: relative to TTT-MoE w/ LB, uTTT-MoE gains $+8.9$ points at 4K and $+1.6$ at 8K but gives back -1.8 at 16K and -0.1 at 32K, and at 32K the strongest fast-weight model is TTT-MoE w/o LB (12.33% vs 10.23% for uTTT-MoE). The aggregate gain at 760M is thus carried by the shorter lengths, and a universal pool does not yet help at the longest context evaluated — the clearest open problem raised here: sharing enlarges the memory a layer can address, but at 32K the binding constraint appears to be what survives in that memory rather than how much is addressable.

outperforms the sliding-window Transformer and the DeltaNet baselines on retrieval at both scales; only full attention, at the cost of a cache that grows with context, scores higher.

The ordering is robust, the attribution is narrower. Aggregate retrieval improves at every step of the ladder, and the ordering survives a six-fold change of scale. That is the direction Section 4.4 predicts. The attribution, however, is narrower than the ordering makes it look: only the final step changes the sharing domain, so universality is one contributor among three rather than the cause of the end-to-end gap. Reading the two adjacent steps as a controlled pair — layer-local pool versus globally shared pool, with routing width and balancing held fixed — is the comparison the framework actually rests on.

Capacity effects surface in retrieval, not in average loss. The state-allocation view of Section 3 predicts that enlarging the addressable memory should change what a model can recover, not how well it predicts on average, and that is the asymmetry the two metrics show at 760M. This is a reason to treat retrieval as the primary probe of fast-weight ownership and per-token loss as a guard against regressions, rather than a reason to prefer whichever metric is more favourable.

The long-context reversal is unexplained. The one result that the framework does not anticipate is the loss of the universal pool's advantage at the longest evaluated context. Section 4.4 argues that sharing raises the state a layer can address; it says nothing about whether the contents of that state survive tens of thousands of tokens of overwriting. If the

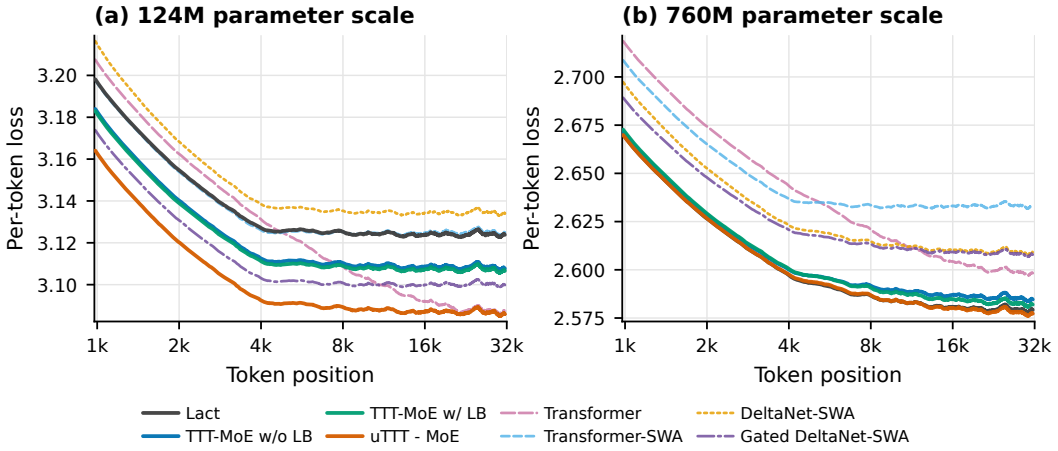

Figure 5: **Per-token validation loss by position** within a 32K-token sequence, at (a) 124M and (b) 760M. Loss on Book-3 against position on a log-scaled axis; positions below 1K are omitted as warm-up, and PTL is the mean of the right-hand end (positions $\geq$ 30K), the horizontal axis of Figure 2. Loss discriminates the models far less than retrieval, and more so with scale: at 124M the per-token-loss ordering matches the RULER ordering (uTTT-MoE lowest at 3.0857), but at 760M the four fast-weight models fall within a 0.0075-nat band (2.5768–2.5843) and the ordering no longer matches — LaCT has the second-best PTL but the worst RULER of the four, while their RULER spread is 4.6 points. Reading either metric alone would support a different conclusion, which is why we report both. uTTT-MoE also reaches lower per-token loss than the full-attention Transformer at both scales (3.0857 vs 3.0869; 2.5768 vs 2.5978) while scoring below it on retrieval: a key-value cache buys recall of specific content, not better average prediction.

binding constraint at 32K is retention rather than addressability, then the update rule and the schedule, not the sharing domain, are the axes that matter there — which is exactly the part of the design space this sweep leaves untested.

6 Experiments: Novel View Synthesis

We next ask whether the ownership conclusions of Section 5 survive a change of modality. Novel view synthesis is a useful second testbed because its read/write structure is explicit rather than inferred: posed views are written into the fast-weight bank and pose-only cameras query it, so rendering quality depends directly on what the shared memory retained.

6.1 Setup

Datasets and metric. We study both object-level and scene-level synthesis and report PSNR on held-out target views. Object models are trained on Objaverse (Deitke et al., 2023) and evaluated on Google Scanned Objects (Downs et al., 2022) at 256×256, with 12 input views and 24 evaluation views per object. Scene models are trained and evaluated on DL3DV (Ling et al., 2024), using its 11K+ training scenes and 140 held-out test scenes at 960×536.

Sequence construction. Each sequence alternates posed-image update chunks with pose-only target-camera query chunks, following Section 4.8. Training randomizes both the order and the number of conditioning views, so the bank must accumulate scene content robustly rather than exploit a fixed view schedule.

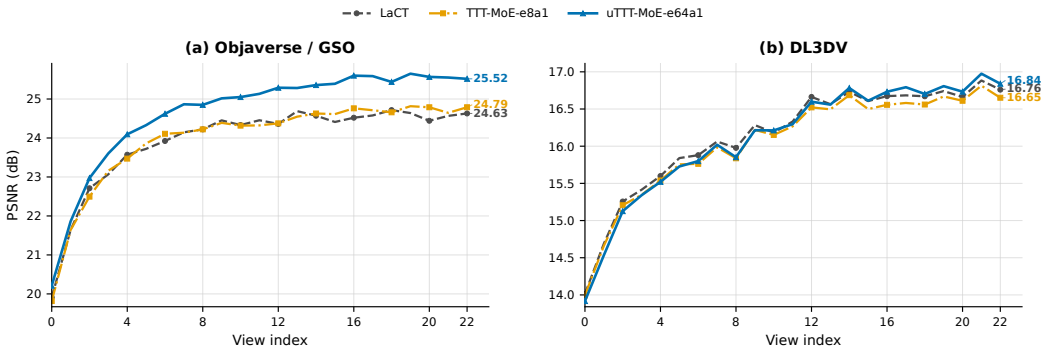

Figure 6: **PSNR against view index** on (a) object-level and (b) scene-level novel view synthesis. Moving right, more posed images have been written into the bank; the value at the right edge is the final PSNR. The three configurations walk the language-modeling ownership ladder collapsed to three rungs — *LaCT* keeps one private dense state per layer, *TTT-MoE-e8a1* a layer-local pool of 8 experts with one active per head, and *uTTT-MoE-e64a1* a global pool of 64 experts with one active per head. The two domains disagree. On Objaverse/GSO the ladder is reproduced: the universal pool separates within a few views and holds to the end, 25.52 dB against 24.79 (layer-local) and 24.63 (layer-private). On DL3DV the curves interleave and the final spread is 0.19 dB (16.84/16.76/16.65), which we do not read as a separation.

Configurations. Figure 6 compares the same three ownership regimes as the language-modeling ladder: a layer-private dense baseline, a layer-local pool of 8 experts per block, and a global pool of 64 experts with one active per token head. With $L = 8$ blocks, the layer-local pool holds $8 \times 8 = 64$ experts in total, matching the global pool's 64; the two therefore have the same total fast-weight capacity and differ only in whether it is partitioned across depth or shared. All three configurations share the same LVSM-style patchification, local attention window, depth, width, optimizer, data schedule, and training-token budget, so a difference is attributable to the fast-weight bank and its routing alone. The main-text comparison fixes these controls; Appendix A reports view-index ablations of dense fast-weight width, stored expert capacity, active expert width at fixed stored capacity, and partitioned pool topology. The sequential schedule is not evaluated, and every routed model reported here uses the aggregated schedule.

Training details. Table 6 (Appendix B) lists the architecture and training settings, identical between the object-level and scene-level models except for the dataset and the per-device batch. Object models are trained for roughly 1.25T visual tokens and scene models for roughly 1.8T, with progressive resolution and view-count scaling, on NVIDIA A100 GPUs with context parallelism at high resolution.

6.2 Results

The object-level result (Figure 6) reproduces the language-modeling ordering across the same three ownership regimes, on a task where the stored content is spatial rather than lexical — the transfer this section set out to test. The scene-level result does not, and we report it rather than the object-level panel alone: a cross-modality claim that rested only on the domain which happened to agree would not be evidence of transfer.

7 Conclusion

TTT models inherit a layout convention from transformers: one fast-weight state per layer, private to that layer. We argued that this convention is arbitrary and studied the universal alternative, in which the whole depth stack reads and writes one shared fast-weight memory. The argument for universality differs by regime. In the dense regime it is a compute

argument: a shared operator lets a fixed fast-weight budget be assembled into one wide state that every layer applies, scaling the FLOPs of the memory path without adding parameters. In the sparse regime it is a capacity and scheduling argument: a global expert pool raises the effective state visible at every forward, lets routing rather than architecture decide how much memory each depth claims, and turns L small per-layer updates into one grouped operation over a single bank. Underneath both, a universal TTT model is a fully recurrent model over the joint (chunk, layer) index, in which depth supplies additional test-time optimization steps on one memory and deep layers can write state that shallow layers read at the next chunk, with layer-private TTT recovered as the fixed-partition special case. Instantiated as uTTT-Dense, TTT-MoE, and uTTT-MoE and compared in language modeling and novel view synthesis, moving fast weights toward a globally shared pool improves long-context retrieval, most clearly at the shorter contexts.

8 Limitations

The empirical support is uneven. The language-modeling comparison of Section 5 is complete at 124M and 760M, and the object-level novel-view-synthesis comparison of Section 6 reproduces its ordering; the scene-level comparison separates the three configurations by less than two tenths of a decibel, so the cross-modality claim rests on one of the two visual domains rather than both. Within the language-modeling evidence, the ownership ordering replicates across both scales on aggregate retrieval, but the per-length decomposition at 760M concentrates the advantage at shorter evaluation lengths, and at 32K the ordering among fast-weight models does not hold; we therefore do not claim that a universal pool improves long-context retrieval uniformly. Two further risks are worth stating. Sharing a bank across depth creates contention, so balancing must be enforced at the pool level and a single depth monopolizing experts is a failure mode with no layer-private analogue. And the sequential schedule strengthens the recurrence but forfeits the grouped execution that motivates part of the design, so quality and throughput need not be optimized by the same configuration. Finally, the scales we reach are modest, and depth-wise sharing interacts with depth, so these conclusions should be re-tested before being extrapolated.

References

Together AI. Long data collections database, 2024. URL `https://huggingface.co/datasets/togethercomputer/Long-Data-Collections`.

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

Aidan Clark, Diego de Las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, et al. Unified scaling laws for routed language models. In *International conference on machine learning*, pp. 4057–4086. PMLR, 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. *arXiv preprint arXiv:2405.21060*, 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*, pp. 13142–13153, 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 *2022 International Conference on Robotics and Automation (ICRA)*, pp. 2553–2560. Ieee, 2022.

Jusen Du, Weigao Sun, Disen Lan, Jiaxi Hu, and Yu Cheng. Mom: Linear sequence modeling with mixture-of-memories. *arXiv preprint arXiv:2502.13685*, 2025.

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.

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. URL https://arxiv.org/abs/2101.00027.

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

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 *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pp. 177–186, 1987.

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

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

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.

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

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

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. *arXiv preprint arXiv:2410.17242*, 2024.

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

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.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Zhixuan Lin, Evgenii Nikishin, Xu He, and Aaron Courville. Forgetting transformer: Softmax attention with a forget gate. In *The Thirteenth 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, et al. 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*, pp. 22160–22169, 2024.

Mehdi SM Sajjadi, Henning Meyer, Etienne Pot, Urs Bergmann, Klaus Greff, Noha Radwan, Suhani Vora, Mario Lučić, Daniel Duckworth, Alexey Dosovitskiy, et al. 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*, pp. 6229–6238, 2022.

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

Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

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. *arXiv preprint arXiv:1701.06538*, 2017.

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

Shawn Tan, Yikang Shen, Zhenfang Chen, Aaron Courville, and Chuang Gan. Sparse universal transformer. In *Proceedings of the 2023 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, et al. End-to-end test-time training for long context. *arXiv preprint arXiv:2512.23675*, 2025.

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

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

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

Qianqian Wang, Yifei Zhang, Aleksander Holynski, Alexei A Efros, and Angjoo Kanazawa. Continuous 3d perception model with persistent state. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 10510–10522, 2025b.

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

Tianyuan Zhang, Sai Bi, Yicong Hong, Kai Zhang, Fujun Luan, Songlin Yang, Kalyan Sunkavalli, William T Freeman, and Hao Tan. Test-time training done right. *arXiv preprint arXiv:2505.23884*, 2025.

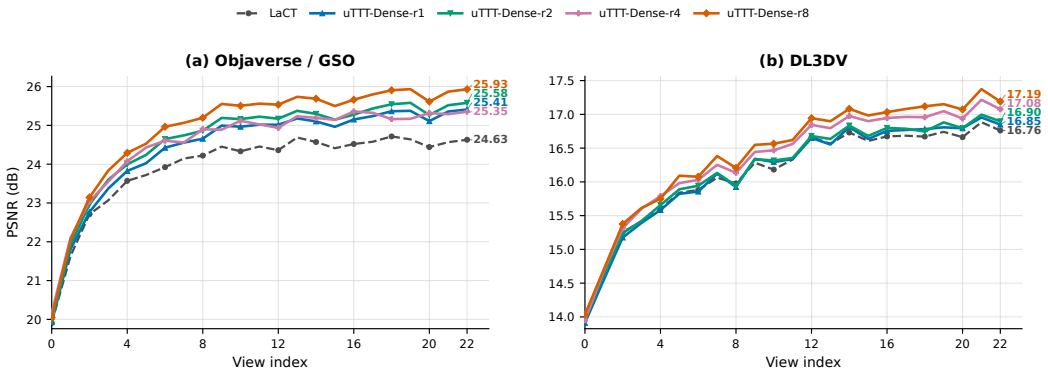

Figure 7: **Dense universal fast weights against view index.** LaCT is the layer-private dense baseline; uTTT-Dense-r1/r2/r4/r8 use one state shared across all eight layers with increasing inner width. The size-matched r8 model finishes highest on both Objaverse/GSO (25.93 vs. 24.63 dB for LaCT) and DL3DV (17.19 vs. 16.76 dB). The intermediate widths are not strictly ordered: on Objaverse/GSO r4 finishes below r2, so the result supports the matched endpoints and the strength of r8 rather than a monotone width law.

A Additional Novel View Synthesis Results

The headline ownership comparison remains in Figure 6 of Section 6. This appendix collects the remaining completed novel-view-synthesis comparisons whose horizontal axis is the view index: dense fast-weight sharing, stored expert capacity, partitioned expert-pool topology, and active expert width. In every panel, moving right means that more posed images have been written into the fast-weight bank; the rightmost value is the PSNR after view 22. Table 7 defines the model suffixes and swept fields. All other architecture, optimizer, data, and training settings follow Table 6.

A.1 Dense Universal Sharing

The dense comparison separates sharing from width. uTTT-Dense-r1 shares one state with the same width and per-layer fast-weight work as one LaCT state, making it FLOPs-matched while using one eighth of LaCT's total fast-weight parameters. At the other endpoint, uTTT-Dense-r8 combines the eight layer-private states into one width-eight shared state: it is size-matched to LaCT while applying eight times as much fast-weight work at every layer. The r2 and r4 variants interpolate between these endpoints.

A.2 Stored Expert Capacity with Top-1 Routing

For the sparse global pool, stored capacity and active compute can be varied separately. We first hold top-1 routing fixed and sweep the total number of stored experts from 8 to 64. Every token head still reads and updates one expert, so this changes the addressable state without increasing the number of active fast-weight operators.

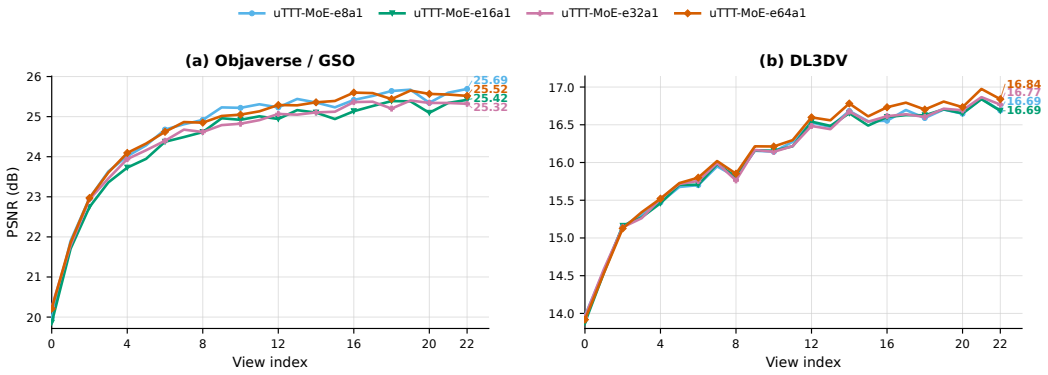

Figure 8: **Stored expert capacity at fixed top-1 access.** All models use one globally shared pool and activate one expert per token head; only the number of stored experts changes. Capacity does not produce a monotone ordering on Objaverse/GSO: e8a1 finishes at 25.69 dB, ahead of e16a1/e32a1/e64a1 at 25.42/25.32/25.52 dB. DL3DV varies over a narrow range, from 16.69 dB for e8a1/e16a1 to 16.77 for e32a1 and 16.84 for e64a1. Merely enlarging the stored pool under fixed top-1 access is therefore weaker than increasing how much of the pool is active in Figure 10.

A.3 Partitioned Pool Topology

We next vary the sharing domain while holding the total stored expert budget and active width fixed. A p1 model exposes one global pool to all eight layers; p2 uses two contiguous four-layer pools; and p4 uses four contiguous two-layer pools. Each pool stores E/P experts, so changing P repartitions the same E experts rather than changing capacity or per-head active compute.

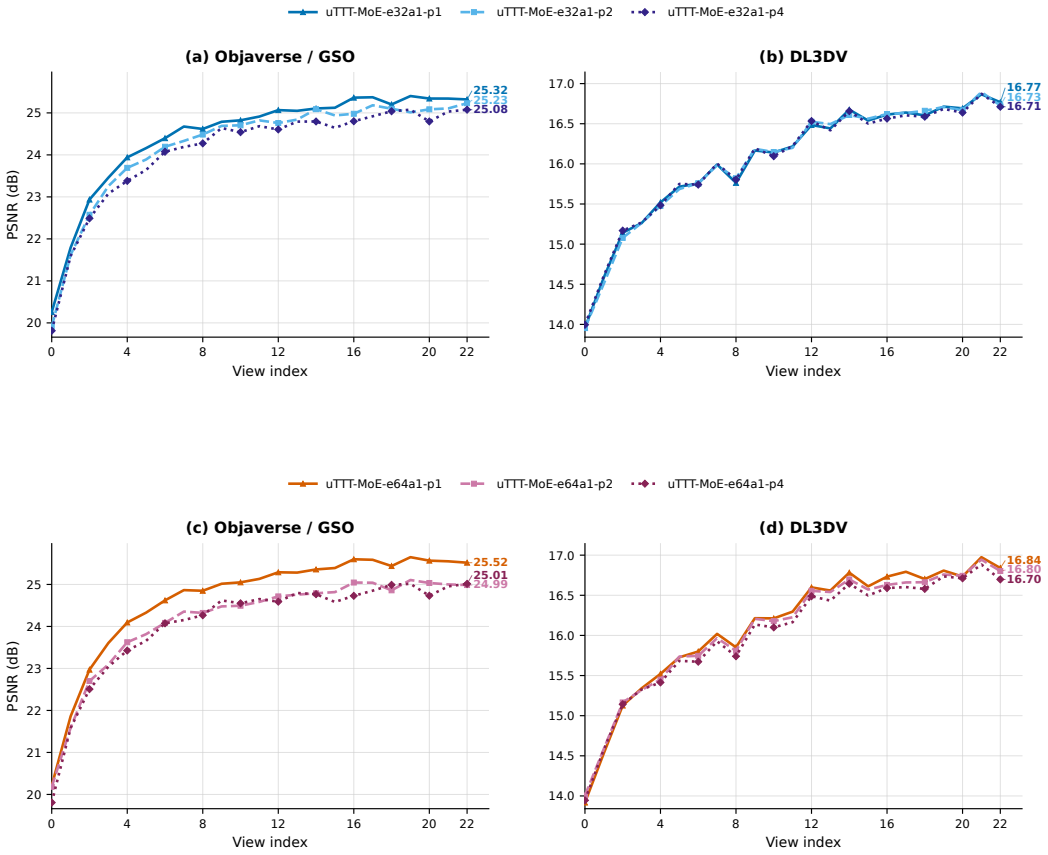

Figure 9: **PSNR as the shared expert pool is partitioned across depth.** The top row fixes $E = 32$ and the bottom row $E = 64$; every model activates one expert per token head. The fully global p1 topology finishes highest in all four dataset–capacity combinations. The separation is clearest on Objaverse/GSO at $E = 64$ (25.52 dB for p1 versus 24.99/25.01 for p2/p4). On DL3DV the differences are small: p1/p2/p4 finish at 16.77/16.73/16.71 dB for $E = 32$ and 16.84/16.80/16.70 dB for $E = 64$. We therefore read this as evidence favoring one global pool on the object-level task, not as a uniform topology effect across domains.

A.4 Active Expert Width

Finally, we keep the stored pool size fixed and increase the number K of experts activated by each token head. Unlike the ownership and topology comparisons, this sweep deliberately raises fast-weight compute with K and should therefore be read as a capacity–compute scaling result, not as an isolated effect of sharing.

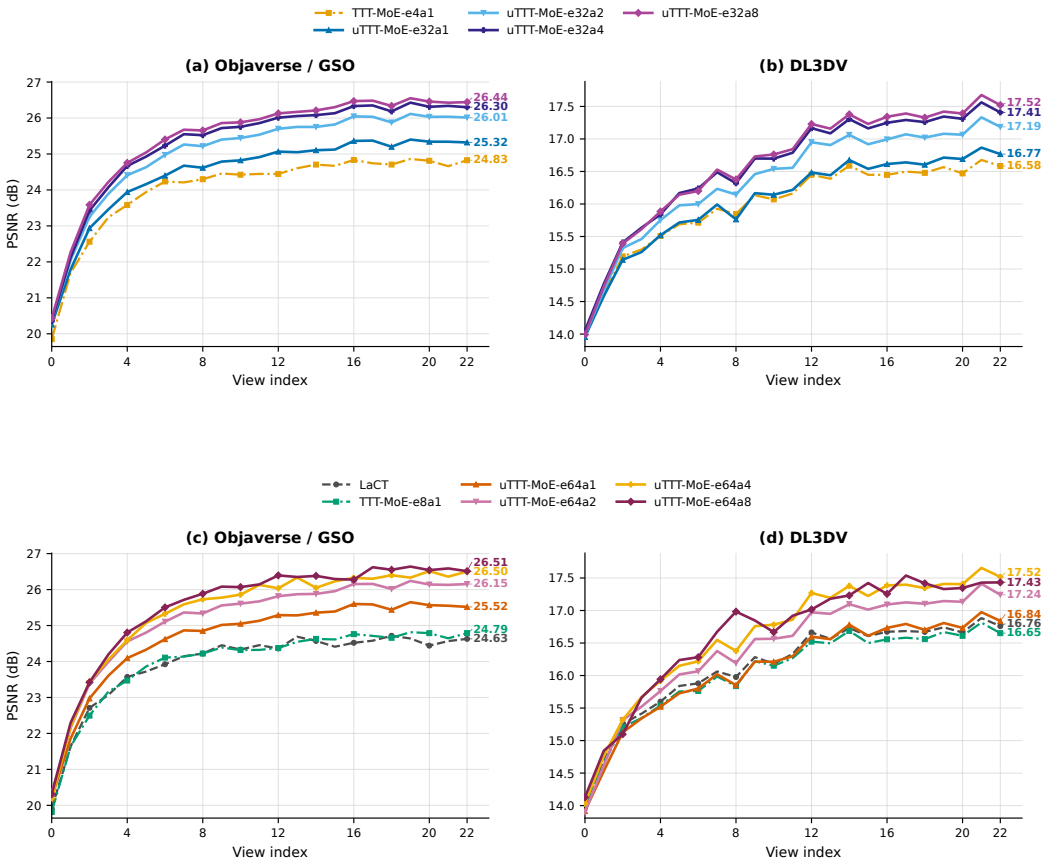

Figure 10: **Active-width scaling against view index.** The top row fixes a 32-expert stored budget and compares the matched layer-local TTT-MoE-e4a1 baseline with uTTT-MoE-e32aK; the bottom row fixes a 64-expert budget and includes LaCT, TTT-MoE-e8a1, and uTTT-MoE-e64aK. Increasing K from 1 to 4 consistently improves the global-pool models. At $E = 32$, a8 finishes highest on both Objaverse/GSO (26.44 dB) and DL3DV (17.52 dB). At $E = 64$, Objaverse/GSO saturates at a4/a8 (26.50/26.51 dB), while DL3DV peaks at a4 (17.52 dB) and falls slightly at a8 (17.43 dB). Thus wider active access helps substantially but is not uniformly monotone at the largest width.

B Model Configurations

Tables 1, 2, 6, and 7 give the architecture and training settings, extracted from the released configuration files. Tables 1 and 6 list the settings shared within each experiment; Tables 2 and 7 summarize the fields that distinguish the compared models; and Tables 3, 4, and 5 give the full hyperparameters of the fast-weight memory and of the attention and linear-attention baselines.

Table 1: Language-modeling configurations. All ownership variants share these settings and differ only in the fast-weight bank.

	124M	760M
Model width d	768	1536
Layers L	12	24
Attention heads	12	24
Head dimension	64	64
FFN	SwiGLU, ratio 4	
Fast-weight heads	4	
Fast-weight update MLP	SwiGLU, low rank 32	
Fast-weight update rule	momentum + Muon, inner-loop LR 1e-3	
Chunk size	4096	
Sliding-window size	4096	
RoPE θ	1e6	
Context length	32768	
Vocabulary	32000	
Optimizer	AdamW (ε = 1e-15)	
Peak learning rate	1e-3	
Schedule	cosine, warmup 1024 steps, min 0.1×	
Gradient clip	1.0	
Precision	bf16	
Batch size / device	4	1
Training steps	10,240	40,960

Table 2: Per-model configuration for the language-modeling comparison, from the released configs. Shared architecture and training settings are in Table 1; this table records what distinguishes each model. All fast-weight models share the fast-weight block: 4 heads, a SwiGLU update MLP of low rank 32, inner-loop learning rate 1e-3, momentum with Muon, reuse-KV memory, and a 4096-token update chunk.

Model	Family	Distinguishing configuration
LaCT	LaCT (dense)	one dense fast-weight state per layer
TTT-MoE w/o LB	LaCT-MoE (local)	layer-local pool $E = 4$, $K = 1$; no load balancing
TTT-MoE w/ LB	LaCT-MoE (local)	layer-local pool $E = 4$, $K = 1$; loss-free balancing, update rate 1e-5
uTTT-MoE	LaCT-MoE (global)	global shared pool, $K = 1$, per-layer router; $E = 48$ (124M) / 96 (760M); loss-free balancing, update rate 3e-5 (124M) / 1e-3 (760M)
Transformer	Transformer	full softmax attention
Transformer-SWA	Transformer	sliding-window attention, window 4096
DeltaNet-SWA	DeltaNet	DeltaNet linear attention mixed with sliding-window attention (window 4096); head dim 64, β delta gate, L2 QK-normalization (Table 5)
Gated DeltaNet-SWA	Gated DeltaNet	gated DeltaNet mixed with sliding-window attention (window 4096); output gate, FFN ratio 3.5 (Table 5)

Table 3: Fast-weight memory configuration for the LaCT-family models. Width, depth, and head count scale with model size (Table 1); where two values appear they are 124M / 760M. TTT-MoE w/o LB is TTT-MoE with balancing disabled.

	LaCT	TTT-MoE	uTTT-MoE
Fast-weight heads	4	4	4
Update MLP	SwiGLU, low rank 32, init gain 0.5		
Inner-loop learning rate	1e-3, per head		
Momentum / Muon	yes / yes		
Learnable TTT scale	yes		
Update chunk	4096		
Memory KV mode	reuse-KV		
Memory QK / feature norm	SiLU QKV	L2	L2
Experts E	1 (dense)	4	48 / 96
Active experts K	—	1	1
Pool mode	—	layer-local	shared (global)
Router	—	—	per-layer softmax
Load balancing	—	loss-free, 1e-5	loss-free, 3e-5 / 1e-3
Aggregate write	—	—	pool-mean, backward full

Table 4: Attention baseline configuration. Width, depth, heads, head dimension 64, RoPE $\theta = 10^6$, SwiGLU FFN (ratio 4), and vocabulary match Table 1.

	Transformer	Transformer-SWA
Attention	full softmax	sliding-window
Window size	∞	4096
Key/value heads	multi-head	multi-head
QKV bias	no	no
QK normalization	no	no

Table 5: Linear-attention baseline configuration. Both mix a DeltaNet recurrence with sliding-window attention; width, depth, and heads match Table 1, with head dimension 64 and RoPE $\theta = 10^6$.

	DeltaNet-SWA	Gated DeltaNet-SWA
Recurrent update	DeltaNet delta rule	gated DeltaNet
Sliding-window size	4096	4096
FFN (SwiGLU) ratio	4	3.5
Short convolution	disabled	disabled
QK activation	SiLU	—
QK normalization	L2	—
β input gate	yes	—
Output gate	no	yes
Output norm	yes	—
Negative eigenvalues	disallowed	disallowed
k / v expansion	1.0 / 1.0	— / 1.0
Computation mode	chunk	chunk

Table 6: Base novel-view-synthesis configuration, shared by the three headline ownership regimes and by the object- and scene-level models (which differ only in dataset and per-device batch). The ablations in Appendix A retain these settings except for the fast-weight field explicitly swept in Table 7.

Model width d	512
Layers L	8
Attention heads	8
Fast-weight heads	8
Head dimension	64
Patch size	8
Fast-weight update MLP	SwiGLU, inner multiplier 1 (headline)
Fast-weight update rule	momentum (no Muon), L2-normalized
Optimizer	AdamW ($\beta = 0.9, 0.95$)
Peak learning rate	4e-4
Schedule	cosine, warmup 2500 steps
Weight decay	0.05
Gradient clip	1.0
Precision	bf16
Training steps	20,000
Router load-balancing α (routed variants)	1e-5

Table 7: Fast-weight configurations in the headline and additional novel-view-synthesis comparisons. In a label *eEaK*, E is the total number of stored experts and K the number activated per token head; pP denotes P contiguous depth pools; and rR is the inner-width multiplier of the single shared dense state. All unswept architecture and optimization settings are those of Table 6.

Comparison	Models	Distinguishing fast-weight configuration
Headline ownership	LaCT; TTT-MoE-e8a1; uTTT-MoE-e64a1	Eight private dense states; eight private 8-expert pools with $K = 1$; or one shared 64-expert pool with $K = 1$. The two MoE variants store 64 experts in total.
Dense sharing	LaCT; uTTT-Dense-r{1,2,4,8}	One dense state shared by all eight layers, with inner multiplier r. The endpoints are FLOPs-matched at $r = 1$ and size-matched at $r = 8$; $r = 2, 4$ interpolate between them.
Top-1 capacity	uTTT-MoE-e{8,16,32,64}a1	One globally shared pool with $K = 1$; the total number E of stored experts is varied.
Active width, $E = 32$	TTT-MoE-e4a1; uTTT-MoE-e32a{1,2,4,8}	32 stored experts: eight private pools of four for the local baseline, or one shared pool with $K \in \{1, 2, 4, 8\}$.
Active width, $E = 64$	LaCT; TTT-MoE-e8a1; uTTT-MoE-e64a{1,2,4,8}	64 stored experts for the routed models: eight private pools of eight for the local baseline, or one shared pool with $K \in \{1, 2, 4, 8\}$.
Pool topology	uTTT-MoE-e{32,64}a1-p{1,2,4}	E and $K = 1$ are fixed within a row. P contiguous pools each span L/P layers and store E/P experts, ranging from one global pool to four two-layer pools.

C Datasets and Licenses

Table 8 lists the external assets used in the experiments and their license or access terms. We do not redistribute raw third-party data; the release manifest records the exact snapshots and access paths.

Table 8: External datasets and evaluation assets, with license or access notes.

Asset	Use in this paper	License or access note
Objaverse (Deitke et al., 2023)	Object-level NVS training source	Dataset-level ODC-By v1.0; individual objects carry Creative Commons licenses recorded in the Objaverse metadata.
Google Scanned Objects (Downs et al., 2022)	Object-level NVS evaluation source	Released under CC-BY 4.0 by Google Research.
DL3DV-10K / DL3DV benchmark (Ling et al., 2024)	Scene-level NVS training and held-out evaluation source	Access is gated through the DL3DV Hugging Face organization and governed by the DL3DV terms of use accepted during dataset access.
Long-Data-Collections (AI, 2024)	Long-context language-model training corpus	Mixed-source text collection; use is governed by the upstream dataset page and constituent-source terms.
Book-3 from The Pile (Gao et al., 2020)	32K per-token loss	The Pile is mixed-source with subset-specific licenses; we record the exact Book-3 snapshot and do not redistribute raw text.
RULER (Hsieh et al., 2024)	Long-context retrieval evaluation	Evaluation code under Apache-2.0; synthetic examples are reproducible from the released scripts.