Last Layer Key-Value Cache is All You Need

Anonymous Author(s)
Affiliation
Address
email

Abstract

The key-value (KV) cache of a long-context Transformer grows with both sequence length and network depth: every layer stores its own keys and values for every retained token. Cross-layer KV sharing shrinks the depth dimension by letting several layers read one cache, but existing designs either preserve quality only at modest sharing factors among adjacent layers, or condense history at every token position, which breaks exact parallel training or requires a redesigned decoder-decoder architecture. We push cross-layer sharing to its limit under a chunked design. LastKV processes each chunk as a standard per-layer-KV Transformer, writes only the last layer's KV cache when the chunk completes, and lets every layer of every future chunk read this single shared cache through ordinary attention. Older history therefore keeps one cache instead of L: the cross-chunk cache drops from $O(TCLd)$ to $O(TCd)$ ($O((T+L)Cd)$ with a nearest-chunk refinement for boundary continuity). Because sharing applies only across completed chunks, within-chunk computation and training parallelism are unchanged: training proceeds chunk by chunk with ordinary backpropagation and no iterative approximation. We separate this cache accounting from the quality question: can all layers actually work from one shared last-layer cache? Across language modeling, novel view synthesis, and autoregressive video generation, controlled comparisons show that maximal cross-layer sharing of older history, with the nearest-chunk refinement enabled for language and video, preserves or improves task quality: it lowers language-model validation loss and stays competitive on RULER retrieval, raises novel-view PSNR, and maintains comparable aggregate VBench scores. Continued pretraining further converts an openly released 8B Transformer into LastKV, which retains one layer of KV out of 32 while scoring like the unconverted baseline across twelve benchmarks.

1 Introduction

Long-sequence modeling increasingly runs into a memory wall. A Transformer retrieves old content by attending to cached keys and values, but the KV cache grows along two axes at once: every retained token keeps a separate cache in every layer [Vaswani et al., 2017, Dai et al., 2019]. For an L-layer model over T chunks of length C, retained history costs $O(TCLd)$ KV memory. Most existing reductions target the token axis, evicting, compressing, or windowing positions [Zhang et al., 2023, Xiao et al., 2024, Li et al., 2024b, Beltagy et al., 2020, Zaheer et al., 2020]. The layer axis admits a different attack: *cross-layer KV sharing*, in which several layers read one cache instead of each keeping their own [Brandon et al., 2024, Zuhri et al., 2025, Wu and Tu, 2024, Sun et al., 2024]. Existing sharing designs, however, stop short of the limit or pay for it elsewhere: sharing among groups of adjacent layers preserves quality only at modest sharing factors, such as the pairwise sharing in CLA's recommended configuration [Brandon et al., 2024, Zuhri et al., 2025], while condensing history onto a single cache restructures attention at every token position: in LCKV, lower layers depend on the top-layer KV of earlier tokens, so exact parallel training is

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

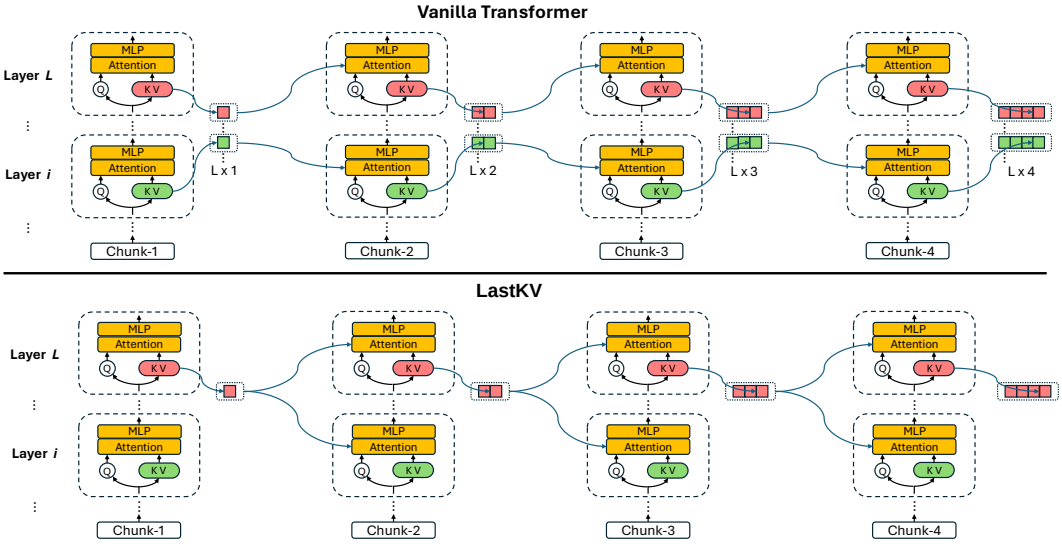


Figure 1: Retained cross-chunk KV cache in a standard chunked Transformer (top) vs. LastKV (bottom). The top panel is an ordinary full-attention Transformer viewed chunk by chunk: chunking only reorganizes the KV-cache accounting and leaves the model and its outputs identical to a standard Transformer. It keeps every layer's KV cache for every past chunk, so the retained cache grows as L times the number of chunks and each layer reads its own per-layer caches: $O(TCLd)$ retained memory. Bottom: LastKV processes each chunk as an unchanged Transformer and, when a chunk completes, appends only its last-layer KV cache to the retained history, which all L layers of every future chunk read: $O(TCd)$. When boundary continuity matters, the nearest previous chunk additionally keeps per-layer caches, α-blended with its last-layer cache, giving $O((T+L)Cd)$.

impossible and an approximate iterative procedure is required [Wu and Tu, 2024], and YOCO avoids this dependency only by redesigning the model into a decoder-decoder whose first half emits the single shared cache [Sun et al., 2024].

This paper pushes cross-layer KV sharing to its limit while leaving the Transformer itself unchanged. The resolution is to separate local computation from long-range memory: sharing does not need to apply at every position, only where the memory lives. We propose LastKV, which partitions a sequence into chunks, processes each chunk with a standard per-layer-KV Transformer, writes only the completed chunk's last-layer KV cache into persistent memory, and lets every layer of every later chunk read that shared cache through ordinary attention. Older context is therefore stored once, not once per layer—the maximal L-fold degree of cross-layer sharing—reducing retained older-history memory from $O(TCLd)$ to $O(TCd)$ while preserving content-addressable retrieval. Because the shared cache is written only when a chunk completes, no layer ever waits on the last-layer output of its own chunk: within-chunk computation stays fully parallel, and training needs no iterative approximation. For language and video settings where the nearest previous chunk supplies immediate boundary continuity, we additionally retain per-layer caches for that single nearest chunk, an $O(LCd)$ overhead confined to one chunk, giving $O((T + L)Cd)$ retained cross-chunk memory rather than $O(TCLd)$: the last layer's KV is all you need for every chunk beyond the immediately preceding one. The model requires no separate hidden state, memory-token bank, or external retrieval store [Rae et al., 2020, Hutchins et al., 2022, Bulatov et al., 2022, Wu et al., 2022].

Figure 1 illustrates the distinction: a standard Transformer carries per-layer KV caches for all history [Vaswani et al., 2017, Dai et al., 2019], while LastKV keeps only the last-layer cache of completed chunks, read by all L layers of later chunks alongside their own local cache—so older history is shared at the full depth L yet stays directly searchable by attention. Chunk size sets how often this compact memory is written: smaller chunks refresh it more frequently, larger chunks rely more on local computation before each write [Katharopoulos et al., 2020, Sun et al., 2023, Gu and Dao, 2023, Hutchins et al., 2022].

We evaluate whether this maximally shared cache preserves useful long-range retrieval in three autoregressive settings: long-context language modeling [Hsieh et al., 2024], novel view synthesis [Mildenhall et al., 2020, Kerbl et al., 2023, Jin et al., 2024, Chen et al., 2024], and autoregressive

video generation [Ho et al., 2022, Peebles and Xie, 2023, Wan Team, 2025]. We first make the cache-memory accounting explicit, then test whether the compact cache remains useful under controlled comparisons with the surrounding training recipe fixed. LastKV lowers language-model per-token validation loss and stays competitive on RULER retrieval, improves PSNR for object- and scene-level novel view synthesis, and maintains comparable aggregate VBench scores in autoregressive video generation [Huang et al., 2024]; beyond training from scratch, continued pretraining converts an openly released 8B Transformer [The Marin Community, 2025] into LastKV while preserving its benchmark scores.

Our contributions are: (1) identifying the chunk boundary as the right place to apply cross-layer KV sharing: sharing one last-layer cache across all L layers removes the layer dimension from retained older-history memory without touching within-chunk attention; (2) instantiating this maximal sharing in a chunk-parallel architecture that trains with ordinary backpropagation (no iterative approximation or decoder redesign), together with an explicit nearest-chunk refinement when boundary continuity is needed; and (3) showing across language — from scratch at 124M–3B and by converting an existing 8B checkpoint — 3D view synthesis, and video generation that a single cross-layer-shared last-layer cache is a viable sequence-modeling primitive rather than a task-specific compression rule.

2 Related Work

2.1 RNNs and Recurrent Sequence Models

RNNs update a hidden state over time, providing a natural temporal inductive bias and enabling intermediate supervision [Elman, 1990]. Gated variants such as LSTMs and GRUs improve optimization and long-horizon credit assignment [Hochreiter and Schmidhuber, 1997, Cho et al., 2014]. Recent recurrent models, including linear attention, RetNet, Mamba, GLA, and DeltaNet, improve efficiency by avoiding dense attention over previous tokens [Katharopoulos et al., 2020, Sun et al., 2023, Gu and Dao, 2023, Yang et al., 2024b,a], but their fixed-size or constrained states may limit fine-grained recall. LastKV keeps a stateful long-context interface, but the retained state is still explicit Transformer KV entries, so older context remains addressable by attention.

2.2 Recurrent Transformers

Several Transformers introduce recurrence to extend context, including Transformer-XL, Compressive Transformer, Universal Transformer, recurrent memory transformer, and block-recurrent Transformers [Dai et al., 2019, Rae et al., 2020, Dehghani et al., 2019, Bulatov et al., 2022, Hutchins et al., 2022]. These works show that recurrence complements attention, but typically reuse hidden activations, compress memory, or add separate memory tokens. LastKV instead applies cross-layer sharing to the retained history of the ordinary KV cache: finished chunks write their last-layer keys and values, and all L layers of later chunks reuse that single compact cache directly. The distinction matters for memory accounting: the goal is not only to shorten the sequence dimension, but to remove the retained layer dimension for older chunks.

2.3 KV-Cache Compression and Streaming

KV-cache eviction, compression, and streaming methods such as H_2O, StreamingLLM, and SnapKV reduce inference-time KV memory for pretrained Transformers by retaining heavy-hitter and recent tokens, preserving attention sinks, or selecting clustered prompt KV positions [Zhang et al., 2023, Xiao et al., 2024, Li et al., 2024b]. These methods are inference policies for an otherwise fixed full-cache attention model, and they compress the token axis of the cache: fewer positions are kept, but every kept position still stores L layer-specific entries. LastKV instead compresses the layer axis, and token selection and quantization can in principle be applied on top of it; our controlled experiments isolate the architectural question of whether one retained layer is enough.

2.4 Cross-Layer KV Sharing

Within a single layer, multi-query and grouped-query attention shrink the KV cache along the head dimension by letting groups of query heads share one KV head [Shazeer, 2019, Ainslie et al., 2023]. Cross-layer sharing compresses the orthogonal depth dimension. CLA computes KV

projections in only a subset of layers and lets each remaining layer attend to the nearest preceding projecting layer's cache, with the recommended configuration sharing each cache between pairs of adjacent layers [Brandon et al., 2024]; MLKV generalizes GQA-style head sharing across groups of consecutive layers [Zuhri et al., 2025]. At the aggressive end, LCKV lets the queries of every layer attend to the keys and values of only the top layer; because each token's lower layers then depend on the top-layer KV of all preceding tokens, exact parallel training becomes impossible, and LCKV trains with an approximate iterative procedure plus a few retained standard-attention layers [Wu and Tu, 2024]. YOCO restructures the model into a decoder-decoder in which a self-decoder with efficient attention spans the first half of the layers and emits one global KV cache consumed by all second-half cross-decoder layers [Sun et al., 2024]. A systematic study unifies CLA-, LCKV-, and YOCO-style configurations and finds that halving the number of KV-caching layers is near-lossless, while more aggressive layer-dimension reductions cost quality, training complexity, or prefill latency [Wu et al., 2025]. Training-free variants merge or share the caches of a pretrained per-layer model post hoc [Liu et al., 2024, Yang et al., 2024c].

All of these designs choose which layers share as a function of depth alone: wherever sharing is active, it applies at every token position. LastKV instead applies maximal cross-layer sharing only where long-range memory lives: within a chunk, every layer keeps its ordinary cache, and only completed chunks are condensed to a single last-layer cache shared by all L layers of future chunks. Within-chunk computation and training therefore remain those of a standard Transformer, with no iterative approximation and no decoder redesign, while the layer dimension is removed from all retained older history. Beyond language decoding, LastKV also extends the evidence for cross-layer sharing to novel view synthesis and autoregressive video generation.

3 LastKV

LastKV keeps long-range history as a single cross-layer-shared cache: the last layer's KV cache of each completed chunk. A local Transformer models within-chunk structure with ordinary per-layer caches, a write step stores only the completed chunk's last-layer keys and values, and later chunks let every layer read this compact history through ordinary attention. The key design choice is therefore not to add a new memory module, but to decide *where* cross-layer sharing applies: within a chunk, every layer keeps its own KV cache as usual; across chunks, all L layers share the one last-layer cache. Two questions have to be answered before this is a usable design. *Which* layer should own the shared cache, and *when* may it be written? The two are coupled: a cache produced by layer j can only be read by layers that run after j on the same tokens, so a design that writes the cache during the forward pass can only share *downward*, from a low layer to the layers above it. That constraint is what forces earlier cross-layer designs either to pick an early source layer [Sun et al., 2024] or to break the dependency with an iterative approximation [Wu and Tu, 2024]. Deferring the write to a chunk boundary removes it: once a chunk has finished all L layers, its last-layer cache is simply data, and every layer of every later chunk may read it. This section develops that mechanism from a general sequence-modeling perspective, independent of any particular modality: we first describe the token-level form to make the cache sharing explicit, then move to the chunk-parallel form used in practice, introduce a recent-chunk aggregation for artificial chunk boundaries, and summarize the resulting cache accounting. Sections 4–7 then instantiate the same mechanism in language model pretraining and continued pretraining, in autoregressive video generation, and in novel view synthesis.

3.1 Preliminaries

Multi-layer Transformer. Consider an L-layer transformer processing a sequence of tokens $\mathbf{x} = (x_1, \ldots, x_N)$. Let $\mathbf{H}^{(0)} \in \mathbb{R}^{N \times d}$ denote the input token embeddings. At layer ℓ, the layer input $\mathbf{H}^{(\ell-1)}$ is projected into queries, keys, and values:

$$\mathbf{Q}^{(\ell)} = \mathbf{H}^{(\ell-1)} W_Q^{(\ell)}, \quad \mathbf{K}^{(\ell)} = \mathbf{H}^{(\ell-1)} W_K^{(\ell)}, \quad \mathbf{V}^{(\ell)} = \mathbf{H}^{(\ell-1)} W_V^{(\ell)}, \tag{1}$$

and multi-head attention reads values according to query-key affinities:

$$\mathrm{Attn}^{(\ell)}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\left(\frac{\mathbf{Q}^{(\ell)} {\mathbf{K}^{(\ell)}}^{\top}}{\sqrt{d_k}} \right) \mathbf{V}^{(\ell)}. \tag{2}$$

Writing $\mathbf{A}^{(\ell)} = \mathrm{Attn}^{(\ell)}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, a standard transformer layer then updates its hidden states with an attention block and an MLP block:

$$\bar{\mathbf{H}}^{(\ell)} = \mathbf{H}^{(\ell-1)} + \mathbf{A}^{(\ell)}, \quad \mathbf{H}^{(\ell)} = \bar{\mathbf{H}}^{(\ell)} + \mathrm{MLP}^{(\ell)}(\bar{\mathbf{H}}^{(\ell)}), \tag{3}$$

where normalization, output projections, and attention masks are omitted for clarity. The KV cache at layer ℓ after processing the sequence is the pair $\mathcal{C}^{(\ell)} = (\mathbf{K}^{(\ell)}, \mathbf{V}^{(\ell)})$.

KV-cache memory. In a chunked long-context setting, suppose the sequence is divided into T retained chunks of length C. A standard Transformer-style history stores the layer-specific cache

$$\mathcal{C}^{\mathrm{full}}_{0:T-1} = \left\{ (\mathbf{K}_t^{(\ell)}, \mathbf{V}_t^{(\ell)}) \mid 0 \le t < T,\ 1 \le \ell \le L \right\}, \tag{4}$$

which scales as $O(TCLd)$ up to the constant factor for storing both keys and values. The full cache is thus a $T \times L$ grid of per-chunk, per-layer entries; cross-layer KV sharing shrinks the layer factor of this grid by letting several layers read the same entry [Brandon et al., 2024, Zuhri et al., 2025, Wu and Tu, 2024, Sun et al., 2024]. If older history is instead represented by a single layer's cache read by all L layers—the maximal degree of sharing—the retained history becomes

$$\mathcal{C}^{\mathrm{last}}_{0:T-1} = \left\{ (\mathbf{K}_t^{(L)}, \mathbf{V}_t^{(L)}) \mid 0 \le t < T \right\}, \tag{5}$$

which scales as $O(TCd)$. The rest of the paper studies whether this maximally shared last-layer cache is sufficient as attention-readable long-range memory.

For some modalities, the nearest previous chunk is not merely distant history: it supplies immediate cross-boundary continuity. In those cases, LastKV keeps the last-layer KV cache for all previous chunks and additionally keeps per-layer caches for only the nearest previous chunk. This two-tier cache has size

$$O(TCd) + O(LCd) = O((T + L)Cd), \tag{6}$$

which is still smaller than the full $O(TCLd)$ cache whenever the retained history spans multiple chunks. We therefore distinguish the core older-history claim, $O(TCd)$, from the practical nearest-chunk refinement, $O((T + L)Cd)$. The refined total improves on the full cache by a factor of $TL/(T + L)$, which approaches the full L-fold saving only when retained history spans many more chunks than layers ($T \gg L$); at moderate T, the nearest-chunk term dominates.

3.2 Token-level LastKV Reuse

To illustrate the idea, let $\mathbf{h}_i^{(\ell)}$, $\mathbf{q}_i^{(\ell)}$, $\mathbf{k}_i^{(\ell)}$, and $\mathbf{v}_i^{(\ell)}$ be the i-th rows of the layer-ℓ hidden, query, key, and value matrices from Section 3.1. The ordinary layer-local projections are

$$\mathbf{q}_i^{(\ell)} = \mathbf{h}_i^{(\ell-1)} W_Q^{(\ell)}, \qquad \mathbf{k}_i^{(\ell)} = \mathbf{h}_i^{(\ell-1)} W_K^{(\ell)}, \qquad \mathbf{v}_i^{(\ell)} = \mathbf{h}_i^{(\ell-1)} W_V^{(\ell)}. \tag{7}$$

After all L layers, token i writes only its last-layer KV pair as retained history:

$$\mathbf{s}_i = \left(\mathbf{k}_i^{(L)}, \mathbf{v}_i^{(L)}\right) = \left(\mathbf{h}_i^{(L-1)} W_K^{(L)},\ \mathbf{h}_i^{(L-1)} W_V^{(L)}\right). \tag{8}$$

Token $i{+}1$ appends this same last-layer pair to every layer's KV input:

$$\tilde{\mathbf{K}}_{i+1}^{(\ell)} = [\mathbf{k}_i^{(L)}; \mathbf{k}_{i+1}^{(\ell)}], \quad \tilde{\mathbf{V}}_{i+1}^{(\ell)} = [\mathbf{v}_i^{(L)}; \mathbf{v}_{i+1}^{(\ell)}], \tag{9}$$

where $[\cdot;\cdot]$ denotes attention-context concatenation, and the read at layer ℓ is

$$\mathbf{o}_{i+1}^{(\ell)} = \mathrm{softmax}\left(\frac{\mathbf{q}_{i+1}^{(\ell)} \left(\tilde{\mathbf{K}}_{i+1}^{(\ell)}\right)^{\top}}{\sqrt{d_k}} \right) \tilde{\mathbf{V}}_{i+1}^{(\ell)}. \tag{10}$$

Thus the old token is stored once, by its last-layer KV cache, and that single cache entry becomes shared attention context for all layers of token $i{+}1$. In this token-level limit, LastKV coincides with the layer-condensed design of LCKV, which lets every layer attend to the top-layer KV of previous tokens [Wu and Tu, 2024]; YOCO similarly condenses history into one shared cache, but produces it at the network midpoint and reads it only with its second-half cross-decoder layers [Sun et al., 2024].

3.3 Chunk-level LastKV Reuse

The token-wise construction above makes the cache-sharing rule explicit, but it is not the form we use in practice, for two reasons. First, the first layer of token $i{+}1$ cannot start until token i finishes layer L, so the dependency chain is sequential in the token dimension; layer-condensed training must break this chain with an iterative approximation and retained standard-attention layers [Wu and Tu, 2024]. Second, even setting parallelism aside, each token's cache would have to be written before the next token can perform its read, exposing a chain of small matrix-vector operations. These operations have low arithmetic intensity and poor accelerator occupancy; on mainstream accelerator devices, they can run roughly 200–300$\times$ below peak FLOPs, creating a severe GPU-utilization bottleneck.

We therefore lift the sharing rule from tokens to chunks. We split the sequence into T chunks of length C, with chunk t containing $x_{tC+1}, \ldots, x_{(t+1)C}$. Each chunk runs a standard L-layer Transformer with full intra-chunk parallelism and no token-wise memory update; only after the whole chunk completes does it write one last-layer KV cache for later chunks. Cross-layer sharing therefore never applies inside the sequence a layer is currently processing, only to completed chunks, so no lower layer ever waits on a last-layer output of its own chunk, and training needs no iterative approximation.

This chunked form keeps the main operators as matrix-matrix computations: projections, local attention, cross-chunk reads, and cache writes operate on token matrices rather than individual token vectors. Stacking token rows gives

$$\begin{aligned}
&\mathbf{H}_t^{(\ell)} = \big[\mathbf{h}_{tC+1}^{(\ell)}; \ldots; \mathbf{h}_{(t+1)C}^{(\ell)}\big], \\
&\mathbf{Q}_t^{(\ell)} = \mathbf{H}_t^{(\ell-1)} W_Q^{(\ell)}, \qquad \mathbf{K}_t^{(\ell)} = \mathbf{H}_t^{(\ell-1)} W_K^{(\ell)}, \qquad \mathbf{V}_t^{(\ell)} = \mathbf{H}_t^{(\ell-1)} W_V^{(\ell)}.
\end{aligned} \tag{11}$$

After chunk t completes all layers, its last-layer KV pair becomes the only retained cache for older history:

$$\mathbf{S}_t = \big(\mathbf{K}_t^{(L)}, \mathbf{V}_t^{(L)}\big) = \big(\mathbf{H}_t^{(L-1)} W_K^{(L)},\ \mathbf{H}_t^{(L-1)} W_V^{(L)}\big). \tag{12}$$

For chunk $t{+}1$, each layer appends the retained cross-chunk context to its own KV input:

$$\tilde{\mathbf{K}}_{t+1}^{(\ell)} = [\mathbf{K}_{\mathrm{ctx}}^{(\ell)}; \mathbf{K}_{t+1}^{(\ell)}], \quad \tilde{\mathbf{V}}_{t+1}^{(\ell)} = [\mathbf{V}_{\mathrm{ctx}}^{(\ell)}; \mathbf{V}_{t+1}^{(\ell)}], \tag{13}$$

where the one-step case uses $(\mathbf{K}_{\mathrm{ctx}}^{(\ell)}, \mathbf{V}_{\mathrm{ctx}}^{(\ell)}) = (\mathbf{K}_t^{(L)}, \mathbf{V}_t^{(L)})$; multi-chunk context is specified below. Since all layers use the same key and value dimensionality, these last-layer KV entries can be concatenated with layer-specific local KV entries and attended to directly. The cross-chunk read is

$$\mathbf{O}_{t+1}^{(\ell)} = \mathrm{softmax}\left(\frac{\mathbf{Q}_{t+1}^{(\ell)} \left(\tilde{\mathbf{K}}_{t+1}^{(\ell)}\right)^{\top}}{\sqrt{d_k}}\right) \tilde{\mathbf{V}}_{t+1}^{(\ell)}. \tag{14}$$

Here $[\cdot;\cdot]$ denotes sequence concatenation, matching Eqs. (9)–(10).

A full-cache chunk history would retain $\{(\mathbf{K}_\tau^{(\ell)}, \mathbf{V}_\tau^{(\ell)})\}_{\tau<t, \ell\le L}$ for later layers. LastKV retains only $\{(\mathbf{K}_\tau^{(L)}, \mathbf{V}_\tau^{(L)})\}_{\tau<t}$. Thus the old-history cache removes the layer dimension, reducing memory from $O(TCLd)$ to $O(TCd)$ before the recent-chunk refinement below. During training, the carried KV tensors remain differentiable: when a later chunk reads $\mathbf{S}_\tau$, its loss backpropagates through the attention path into the last-layer keys and values written by chunk τ. We do not detach retained caches or truncate gradients across chunk boundaries. The sequential dependency is only across chunks; each chunk's intra-chunk Transformer pass remains fully parallel.

Implementation recipe. For a new chunk, the model first builds the retained context from older chunks, concatenates that context with the chunk-local KV cache at each layer, and computes ordinary attention. After all L layers finish, only the chunk's last-layer KV pair is committed to persistent memory. The current chunk's activations and local per-layer KV caches are temporary computation buffers; the retained memory across older chunks is the last-layer cache described above, optionally plus the nearest-chunk refinement in Section 3.4. The first chunk is a special case only in that it has no history to read: it runs as a plain Transformer and its last-layer cache seeds $\hat{\mathcal{R}}$.

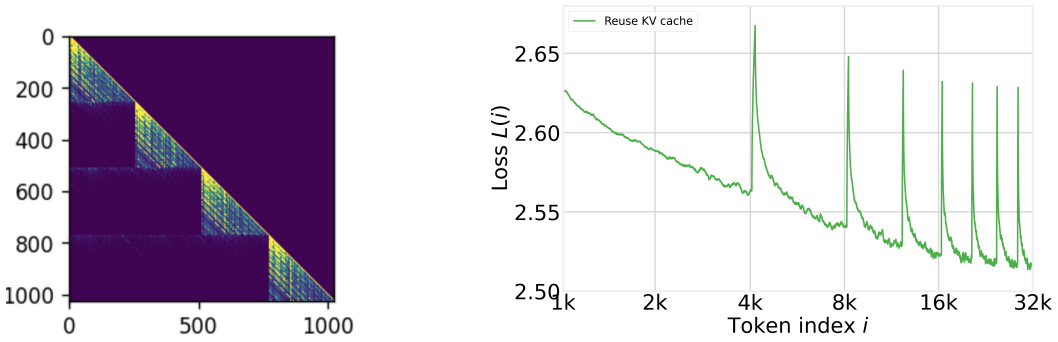
What the chunk boundary costs. Deferring the write buys the sharing direction, and it is not free. Three costs are worth naming. *Serialization*: chunks must be processed in order, so the sequence dimension is parallel only within a chunk — with C tokens per chunk, a N-token sequence exposes C-wide parallelism N/C times instead of N-wide parallelism once. *Repeated reads*: each chunk attends over the whole retained history, so a naive implementation touches $\Theta(T^2Cd)$ retained entries over a sequence of T chunks, where a single fused attention pass would touch each entry once. *Staleness*: within a chunk, tokens see history only up to the previous boundary, so the effective context is quantized to multiples of C — the artifact that Section 3.4 addresses. Chunk size is the dial for all three: larger C means fewer, cheaper boundaries but more staleness and a larger nearest-chunk term in Eq. (6). Section 5 (Table 2) measures the throughput of an optimized chunk loop at 8B scale.

Decoding. At inference the same loop runs with the inner pass switched from parallel to token-by-token. Prefill is unchanged from a standard Transformer within each chunk. During decode, a step attends over the shared history $\hat{\mathcal{R}}$, the nearest-chunk buffers, and the tokens generated so far in the current chunk; nothing is recomputed at a boundary, since the last-layer entries needed there were already produced by the forward pass that emitted those tokens. The retained state therefore grows by one shared entry per token rather than L, which is the whole point of the design, and the per-step attention cost falls in proportion — a decode step reads $O((T + L)Cd)$ cached entries instead of $O(TCLd)$.

Why the last layer. Given that the write is deferred, any layer's cache is now a legal choice, and the last layer is one end of that range. Two arguments favor it. It is the deepest summary available: the last layer's keys and values are computed from a representation that has passed through every block, so a query issued by *any* layer of a later chunk is matched against the most processed form of the history. And it is the only choice that costs nothing to produce — it is already computed by the ordinary forward pass, so *Reuse KV* adds no parameters and no FLOPs, whereas producing a cache at an intermediate depth for all layers to read means either committing to that depth or learning extra projections. Neither argument is decisive on its own, so we treat the source layer as an empirical question and sweep it in both language modeling (Figure 6) and novel view synthesis (Figure 12). Deeper sources are consistently stronger and the last layer is at or near the best, but the choice is not knife-edge: a band of the deeper layers performs comparably, and lower sources remain usable — every source gap still works in NVS, and in language modeling only the lowest sources fall away, and only on long-context retrieval. That lower sources also work leaves design room: since the shared memory need not be produced at the very top of the stack, a shallow recurrent sub-network could generate it while the deeper layers stay feed-forward — a small internal recurrent module rather than a fully recurrent model. We use the last layer here for its zero-cost *Reuse KV* and leave that direction to future work.

3.4 Nearest-Chunk KV Refinement

For novel view synthesis, views are largely independent chunks. For language modeling and video generation, however, the nearest previous chunk supplies immediate cross-boundary continuity, and boundary-only cache writes can introduce artifacts; language modeling makes the failure mode easiest to visualize, as illustrated in Figure 2.



(a) Attention map.

(b) Per-token loss.

Figure 2: Language-modeling chunk-boundary artifacts from compact last-layer KV reuse. After crossing a chunk boundary, the attention map becomes block-like: tokens in the next chunk attend within the current chunk and fail to read the full previous chunk. The loss curve shows matching spikes at chunk transitions, confirming a discontinuity introduced by boundary-only cache writes.

Figure 2(a) shows the resulting failure mode in the attention map: after crossing an artificial chunk boundary, tokens in the next chunk attend mainly within the current chunk and fail to read the full previous chunk content. Figure 2(b) shows matching spikes in per-token loss at the same transitions. The most recent previous chunk $t-1$ is therefore treated as a special short-range context. At layer ℓ, we combine the layer-ℓ KV cache from chunk $t-1$ with its last-layer KV state using an interpolation coefficient $\alpha \in [0, 1]$:

$$\mathbf{K}^{(\ell)}_{\mathrm{ctx},\,t-1} = \alpha\,\mathbf{K}^{(\ell)}_{t-1} + (1-\alpha)\,\mathbf{K}^{(L)}_{t-1}, \quad \mathbf{V}^{(\ell)}_{\mathrm{ctx},\,t-1} = \alpha\,\mathbf{V}^{(\ell)}_{t-1} + (1-\alpha)\,\mathbf{V}^{(L)}_{t-1}. \tag{15}$$

In our experiments, we set $\alpha = 0.5$. This blend is important for temporal tasks such as video generation and language modeling: the layer-specific term preserves immediate cross-boundary continuity, while the last-layer term keeps the shared compact cache. For older chunks $\tau \in \{0, \ldots, t-2\}$, we do not keep layer-specific caches; each layer reads only the shared last-layer KV state:

$$\mathbf{K}^{(\ell)}_{\mathrm{ctx},\,\tau} = \mathbf{K}^{(L)}_{\tau}, \quad \mathbf{V}^{(\ell)}_{\mathrm{ctx},\,\tau} = \mathbf{V}^{(L)}_{\tau}, \quad \forall\, \ell \in \{1, \ldots, L\}. \tag{16}$$

The full cross-chunk context at layer ℓ for chunk t concatenates the last-layer KV from chunks 0 through $t-2$ with the refined context from chunk $t-1$:

$$(\mathbf{K}^{(\ell)}_{\mathrm{ctx}},\, \mathbf{V}^{(\ell)}_{\mathrm{ctx}}) = \mathrm{concat}\big(\{(\mathbf{K}^{(L)}_{\tau}, \mathbf{V}^{(L)}_{\tau})\}_{\tau=0}^{t-2}, (\mathbf{K}^{(\ell)}_{\mathrm{ctx},\,t-1}, \mathbf{V}^{(\ell)}_{\mathrm{ctx},\,t-1})\big). \tag{17}$$

This two-tier scheme gives the nearest previous chunk access to both same-layer detail and the last-layer shared cache, while chunks 0 through $t-2$ are represented only by their last-layer KV. It therefore preserves the immediate predecessor more faithfully while keeping the cross-chunk memory cost at $O((T + L) \cdot C \cdot d)$ rather than $O(T \cdot C \cdot L \cdot d)$.

3.5 Cache Accounting and Streaming Inference

Table 1 makes the retained-cache accounting explicit as a function of chunks T, chunk length C, layers L, and hidden width d, and Algorithm 1 gives the single chunk loop that realizes this accounting in both training and streaming inference. The remaining question is empirical: once all layers share one last-layer cache for older history, does the resulting model still preserve useful long-range information? Sections 4–7 instantiate LastKV in each domain and evaluate that quality question under matched training recipes.

4 LLM Pretraining

Language modeling is where the retained-cache question is most pressing, for two reasons. First, language models underlie agentic systems that interleave reasoning with tool use [Yao et al., 2023], and realistic agentic tasks unroll into long trajectories—resolving one software issue means reading and editing a codebase over many steps [Jimenez et al., 2024]. The whole trajectory stays in the KV cache, so cross-layer sharing removes a factor of L from exactly the memory that grows with trajectory

Table 1: Retained cross-chunk KV-cache accounting. Counts omit the constant factor for storing both keys and values and exclude temporary current-chunk activations. The core LastKV claim is the older-history row; the practical nearest-chunk refinement additionally keeps the per-layer cache of only the nearest chunk to smooth chunk boundaries.

Design	Retained cache for previous chunks	Asymptotic memory	Relative to full cache
Full per-layer KV cache	T chunks $\times$ L layer-specific KV caches	$O(TCLd)$	1
LastKV older-history cache	T chunks $\times$ one last-layer KV cache	$O(TCd)$	$1/L$
LastKV with nearest-chunk refinement	last-layer KV for T chunks plus per-layer KV for the single nearest chunk	$O((T+L)Cd)$	$(T+L)/(TL)$

Algorithm 1 LastKV: one chunk loop for training and streaming inference

Require: chunk size C; L layers with projections $W_Q^{(\ell)}, W_K^{(\ell)}, W_V^{(\ell)}, W_O^{(\ell)}$; per-head blend coefficients $\bar{\alpha}^{(\ell)}$

$\hat{\mathcal{R}} \leftarrow \emptyset$ ▷ retained history: *one* shared last-layer cache
$\mathbf{P}^{(\ell)}, \mathbf{C}^{(\ell)} \leftarrow \emptyset$ for all ℓ ▷ nearest-chunk and current-chunk per-layer caches
for each chunk $t = 0, 1, \ldots$ **do** ▷ sequential across chunks only
  for $\ell = 1$ **to** L **do** ▷ ordinary Transformer layer, unchanged
    project the layer input to $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ and append $(\mathbf{K}, \mathbf{V})$ to $\mathbf{C}^{(\ell)}$
    $\mathbf{O} \leftarrow \mathrm{Attn}\big(\mathbf{Q},\ [\hat{\mathcal{R}};\ \mathrm{blend}_{\bar{\alpha}^{(\ell)}}(\mathbf{P}^{(\ell)}, \mathbf{P}^{(L)});\ \mathbf{C}^{(\ell)}]\big)$ ▷ Eqs. (15)–(17)
    apply the output projection and MLP as usual
  end for
  — chunk boundary: commit the last layer, roll the buffers —
  $\hat{\mathcal{R}} \leftarrow [\hat{\mathcal{R}};\ \mathbf{P}^{(L)}]$ ▷ chunk $t-1$ graduates to history; its other $L-1$ layers are freed
  $\mathbf{P}^{(\ell)} \leftarrow \mathbf{C}^{(\ell)}$; $\mathbf{C}^{(\ell)} \leftarrow \emptyset$ for all ℓ
end for

Training vs. inference. The two modes differ only in how the inner loop consumes a chunk: training processes all C tokens in parallel and backpropagates through the appended caches (no detaching, so $\mathbf{K}^{(L)}, \mathbf{V}^{(L)}$ receive gradients from every later chunk); streaming inference prefills in parallel and then decodes token by token. The boundary step is identical.

Retained state. $|\hat{\mathcal{R}}| \le TCd$ (one copy, not L) + $|\{\mathbf{P}^{(\ell)}\}| = LCd + |\{\mathbf{C}^{(\ell)}\}| \le LCd$, i.e. $O((T+L)Cd)$ versus $O(TCLd)$ for a full per-layer cache (Table 1). The chunk-parallel *training* implementation instead materializes per-layer views of $\hat{\mathcal{R}}$ for kernel simplicity.

length. Second, depth is a lever that models keep pulling. Deeper Transformers generalize more compositionally than shallower ones at matched parameter count [Petty et al., 2024], and recent work scales *virtual* depth by looping a recurrent block at test time rather than adding parameters [Dehghani et al., 2019, Geiping et al., 2025]. Under either kind of depth — more layers or more loop iterations — a design that retains one layer's KV cache instead of every layer's becomes more valuable, since the saving grows with the depth being added. Cross-layer KV sharing is the established response to this pressure [Sun et al., 2024, Brandon et al., 2024, Wu and Tu, 2024, Zuhri et al., 2025], but prior designs share *downward*: upper layers read a cache produced by a lower layer, because a lower layer cannot wait on a value the network has not computed yet. Chunked recurrence removes that constraint — the shared cache is written when a chunk completes, so every layer of a later chunk can read the *last* layer's cache — and the experiments below show that this upward direction works well in practice.

These settings are also where the design is most at risk. Retrieval from context is precisely what a KV cache exists for, and LastKV keeps one layer of it out of L, so the first-order concern is that in-context retrieval degrades. For this reason we make long-context language-model pretraining the first and most heavily instrumented experiment in the paper: models are pretrained from scratch at 32K sequence length, and the evaluation targets whether long context still works — per-token validation loss as a function of position, which reveals whether distant tokens are being used at all, and per-task RULER retrieval, which probes the retrieval ability the compact cache most directly threatens.

4.1 Method

Section 3 defines the chunk loop in the abstract; this subsection gives the language-model instantiation, which is the form all of our LM results use. The shared-cache flow is the schematic of Figure 1, and Algorithm 2 gives the training forward.

Chunking and local attention. Tokens are partitioned into chunks, where chunk t contains $x_{tC+1}, \ldots, x_{(t+1)C}$; we use $C = 4096$ throughout unless stated otherwise. Each chunk is processed with causal gated self-attention [Qiu et al., 2025], and no cross-chunk cache is written until the chunk is complete. Let $\mathbf{A}_t^{(\ell,m)}$ denote the causal SDPA output of head m before the output projection; we use

$$\widehat{\mathbf{A}}_t^{(\ell,m)} = \mathbf{A}_t^{(\ell,m)} \odot \sigma\Big(\mathbf{H}_t^{(\ell-1)} W_G^{(\ell,m)}\Big), \quad \widehat{\mathbf{A}}_t^{(\ell)} = \mathrm{Concat}_{m=1}^{M}\Big(\widehat{\mathbf{A}}_t^{(\ell,m)}\Big) W_O^{(\ell)}. \tag{18}$$

The gate is applied identically in the baseline, so it is not part of the treatment.

Write, read, and free. After chunk t is complete, its last-layer KV pair $\mathbf{S}_t = (\mathbf{K}_t^{(L)}, \mathbf{V}_t^{(L)})$ is written to memory and the other $L-1$ layers' caches for that chunk are released. When processing chunk $t+1$, each layer forms the cross-chunk context from past last-layer caches, appends it to the current chunk KV input, and reads with ordinary attention as in Eqs. (13)–(17). Past context is fully visible; the current chunk remains causally masked. Nothing else about the block changes — no extra normalization, no memory module, no separate read/write projections.

Positions. Rotary embeddings [Su et al., 2024b] are applied before caching, with the offset of a chunk set to the number of retained plus nearest-chunk entries preceding it, so a retained entry keeps the absolute position it had when it was written. Retained history is therefore addressed by the same positional scheme as local context, and no re-rotation is needed at a boundary.

The blend coefficient. The nearest-chunk interpolation of Eq. (15) is learned rather than fixed: $\bar{\alpha}^{(\ell)} = \sigma(a^{(\ell)})$ with a per-layer, per-head logit $a^{(\ell)} \in \mathbb{R}^H$ initialized to 0, so training starts at $\alpha = 0.5$ and each head can move toward its own layer's detail or toward the shared cache. This is the only parameter LastKV adds — $L \cdot H$ scalars, 576 at 760M, five orders of magnitude below the model size — which is what makes the comparison against the baseline iso-parameter.

4.2 Experiment Setup

Datasets and metrics. We train on Long-Data-Collections [Together AI, 2024], drawing up to 60B tokens from the 68.8B-token corpus, and tokenize with the Mistral tokenizer [Jiang et al., 2023]. We report two metrics. The first is *per-token validation loss* on Books3 [Gao et al., 2020] over 32K-token sequences, evaluated for 10,240 steps: plotting loss against position within the context tests whether a model actually uses the context it is given, since a loss that keeps decreasing with position indicates that distant tokens are being exploited while a plateau indicates that they are not. The second is *retrieval accuracy* on RULER [Hsieh et al., 2024], reported per task at 4K, 8K, 16K, and 32K so that task families — single-needle, multi-key, multi-value, aggregation, and QA — can be read separately rather than through one average.

Model and training details. All models use a 32K-token training sequence length, a RoPE [Su et al., 2024b] base of 1M, and the gated attention block of Eq. (18). LastKV additionally uses 4K-token chunks and the nearest-chunk refinement of Section 3.4 initialized at $\alpha = 0.5$. We train three scales:

- **124M**: 10B tokens, global batch size 32, 8 A100 80G GPUs.
- **760M**: 24 blocks, model dimension 1536; 40B tokens (40,960 steps), global batch size 32, 32 A100 80G GPUs.
- **3B**: 60B tokens (30,720 steps), global batch size 64, 64 A100 80G GPUs.

Full architecture specifications and optimizer schedules are in Tables 9 and 10.

Algorithm 2 LastKV: chunk-parallel forward with last-layer KV write (language modeling)

Require: chunk embeddings $\mathbf{H}_t^{(0)} \in \mathbb{R}^{B\times C\times D}$ for chunks $t = 0, \ldots, T-1$; L layers with projections $W_Q^{(\ell)}, W_K^{(\ell)}, W_V^{(\ell)}, W_G^{(\ell)}, W_O^{(\ell)}$; per-head blend logits $a^{(\ell)} \in \mathbb{R}^H$ (learnable, init 0)

$\mathcal{R}_K^{(\ell)}, \mathcal{R}_V^{(\ell)} \leftarrow \emptyset$ for all ℓ ▷ retained history: last-layer KV of chunks $0..t-2$

$\mathbf{P}_K^{(\ell)}, \mathbf{P}_V^{(\ell)} \leftarrow \emptyset$ for all ℓ ▷ nearest chunk $t-1$: per-layer KV, $\mathbb{R}^{B\times C\times H\times d_h}$

for $t = 0$ **to** $T-1$ **do** ▷ sequential over chunks; fully parallel within a chunk

  $\mathbf{X} \leftarrow \mathbf{H}_t^{(0)}$

  for $\ell = 1$ **to** L **do**

    $\bar{\mathbf{X}} \leftarrow \mathrm{RMSNorm}(\mathbf{X})$

    $\mathbf{Q}, \mathbf{G} \leftarrow \mathrm{split}(\bar{\mathbf{X}} W_Q^{(\ell)})$; $\mathbf{K} \leftarrow \bar{\mathbf{X}} W_K^{(\ell)}$; $\mathbf{V} \leftarrow \bar{\mathbf{X}} W_V^{(\ell)}$ ▷ $\mathbb{R}^{B\times C\times H\times d_h}$

    $(\mathbf{Q}, \mathbf{K}) \leftarrow \mathrm{RoPE}\big(\mathbf{Q}, \mathbf{K};\ \mathrm{offset} = |\mathcal{R}_K^{(\ell)}| + |\mathbf{P}_K^{(\ell)}|\big)$ ▷ keys cached post-RoPE

    if $t \geq 1$ **then** ▷ $\bar{\alpha}^{(\ell)} = \sigma(a^{(\ell)}) \in \mathbb{R}^H$: per-head blend, 0.5 at init (Eq. 15)

      $\tilde{\mathbf{K}} \leftarrow \big[\mathcal{R}_K^{(\ell)};\ \bar{\alpha}^{(\ell)} \odot \mathbf{K}_{t-1}^{(L)} + (1-\bar{\alpha}^{(\ell)}) \odot \mathbf{P}_K^{(\ell)};\ \mathbf{K}\big]$

      $\tilde{\mathbf{V}} \leftarrow \big[\mathcal{R}_V^{(\ell)};\ \bar{\alpha}^{(\ell)} \odot \mathbf{V}_{t-1}^{(L)} + (1-\bar{\alpha}^{(\ell)}) \odot \mathbf{P}_V^{(\ell)};\ \mathbf{V}\big]$

    else

      $\tilde{\mathbf{K}} \leftarrow \mathbf{K}$; $\tilde{\mathbf{V}} \leftarrow \mathbf{V}$

    end if

    $\mathbf{O} \leftarrow \mathrm{FlashAttn}(\mathbf{Q}, \tilde{\mathbf{K}}, \tilde{\mathbf{V}};\ \mathrm{causal})$ ▷ past context fully visible; current chunk causal

    $\mathbf{O} \leftarrow \mathbf{O} \odot \sigma(\mathbf{G})$ ▷ elementwise output gate (Eq. 18)

    $\mathbf{X} \leftarrow \mathbf{X} + \mathbf{O} W_O^{(\ell)}$; $\mathbf{X} \leftarrow \mathbf{X} + \mathrm{MLP}^{(\ell)}(\mathrm{RMSNorm}(\mathbf{X}))$

    $(\mathbf{K}_t^{(\ell)}, \mathbf{V}_t^{(\ell)}) \leftarrow (\mathbf{K}, \mathbf{V})$ ▷ layer-ℓ chunk KV; kept only until the next boundary

  end for

  $\mathbf{H}_t^{(L)} \leftarrow \mathbf{X}$ ▷ to LM head / loss

  — chunk boundary: commit last-layer KV, roll nearest-chunk buffers —

  for $\ell = 1$ **to** L **do**

    if $t \geq 1$ **then**

      $\mathcal{R}_K^{(\ell)} \leftarrow [\mathcal{R}_K^{(\ell)};\ \mathbf{K}_{t-1}^{(L)}]$; $\mathcal{R}_V^{(\ell)} \leftarrow [\mathcal{R}_V^{(\ell)};\ \mathbf{V}_{t-1}^{(L)}]$ ▷ chunk $t-1$ graduates to history

    end if

    $(\mathbf{P}_K^{(\ell)}, \mathbf{P}_V^{(\ell)}) \leftarrow (\mathbf{K}_t^{(\ell)}, \mathbf{V}_t^{(\ell)})$

  end for

end for

Note (differentiability): appends write into preallocated storage through a custom autograd function whose backward routes gradients to both the cache prefix and the appended block; retained $\mathbf{K}^{(L)}, \mathbf{V}^{(L)}$ receive gradients from all future chunks and are never detached.

Note (memory): $\mathcal{R}^{(\ell)}$ holds identical values for every ℓ (the last layer's entries); it is logically one shared cache of size $O(TCd)$. The chunk-parallel training implementation keeps per-layer views for kernel simplicity; inference stores one copy (Alg. 1).

Baselines. The comparison at each scale is against a full-cache Transformer that shares the identical gated-attention block, dimensions, tokenizer, data order, optimizer, and schedule; the retained-cache scheme is the only variable that changes. Since LastKV adds only $L \cdot H$ blending scalars (Section 4.1), the comparison is iso-parameter and iso-recipe. At 760M we additionally run the three KV-construction modes of Section 3 — Reuse KV, Shared KV, and Per-layer KV — so that the choice of how the retained cache is built is separated from the choice of which layer supplies it; the 124M and 3B runs use Reuse KV, our default.

4.3 Main Results

We report each scale as one grid of per-task RULER accuracy plus the two loss curves: Figure 3 at 760M, where all three KV-construction modes were run, and Figures 4 and 5 at 124M and 3B, where we compare Reuse KV against the full-cache baseline.

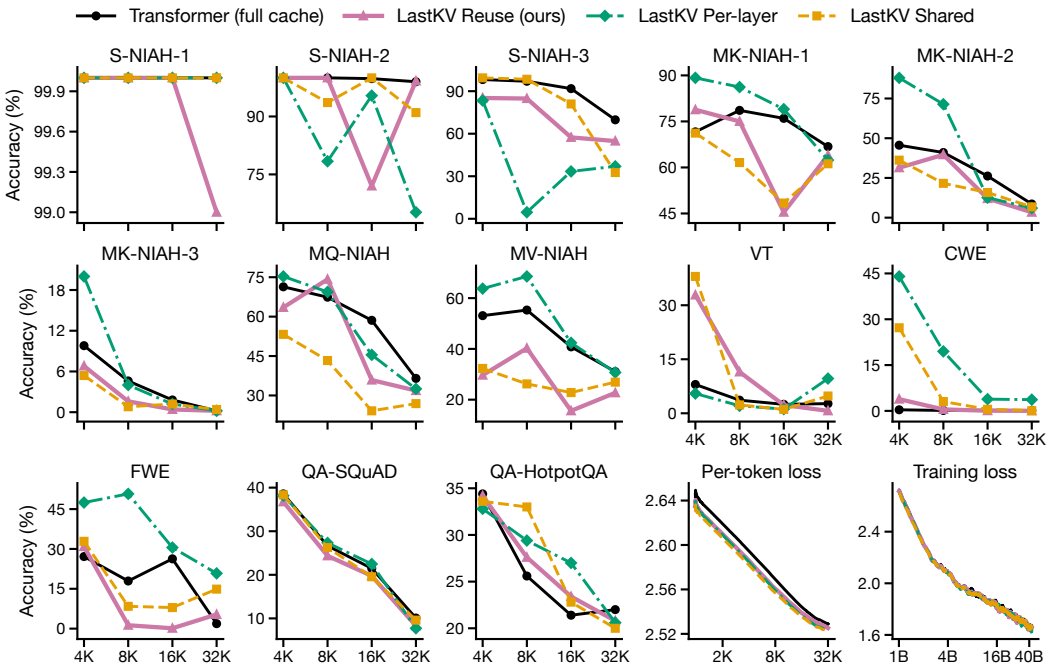


Figure 3: 760M results with gated attention, comparing the full-cache Transformer with the three LastKV KV-construction modes (our language models use Reuse KV). First 13 panels: per-task RULER accuracy at context lengths 4K–32K (higher is better; each panel uses its own accuracy range); Table 11 reports the same values as numbers. Last two panels (lower is better): per-token validation loss on Books3 over a log-scale token index (time-weighted EMA smoothing 0.99), and smoothed training loss over log-scale training tokens.

Per-Token Loss. The per-token loss panels of Figures 3–5 show that LastKV lowers per-token validation loss relative to the Transformer baseline throughout the 32K-token context at all three scales, with the widest margin at 124M. The curves keep decreasing with position rather than flattening, which is the signature of context actually being used. The adjacent training-loss panels show the two training curves tracking each other closely, with LastKV ending slightly lower at 124M and 3B and level at 760M — training as a recurrent model costs nothing in optimization. That both training and validation loss come out at or below the full-cache baseline, at matched parameters and essentially matched per-token compute, admits a natural reading: chunk recurrence improves *token efficiency* — more is learned from each training token — and therefore delivers a *compute-equivalent gain*, lower loss for the same training compute. A mechanism consistent with this is the virtual-depth view of Section 5: the chunk loop re-applies the full stack over an evolving recurrent state, so each token's prediction draws on computation accumulated across the entire prefix rather than only on its own forward pass. Gains also appear in the initial region; this is consistent with the compact-cache training recipe affecting within-chunk computation as well, although isolating that effect requires a separate training-recipe ablation.

RULER. The figures report all 13 RULER tasks at context lengths from 4K to 32K; Tables 11 and 12 in the appendix give the 760M numbers with and without gated attention. At 760M with the gate, retrieval-heavy multi-key tasks favor the full cache at long contexts while single-needle, QA, and aggregation tasks are preserved or improved under the compact cache; without the gate, the LastKV variants overtake the full-cache baseline in the 13-task mean at 8K–32K. At 124M the split runs the other way on most tasks: LastKV leads on the multi-key, multi-value, word-extraction, and QA tasks — by a wide margin on FWE, where the baseline collapses to near zero — while the full cache retains its advantage on the single-needle tasks. At 3B the division sharpens into task families: the full cache dominates the multi-key, multi-value, and variable-tracking tasks, while LastKV matches or exceeds it on single-needle retrieval and leads clearly on the aggregation (CWE, FWE) and QA tasks. Across scales the compact cache is therefore not uniformly better or worse; which task families it favors depends on the scale and on the gate.

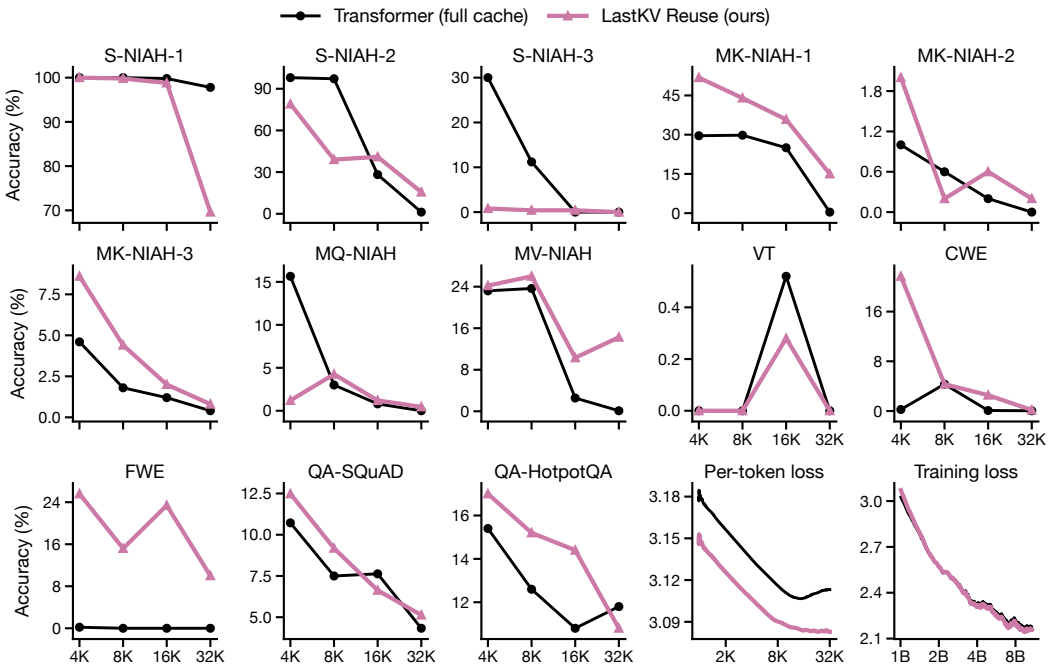


Figure 4: 124M results with gated attention, laid out as in Figure 3: per-task RULER accuracy at context lengths 4K–32K in the first 13 panels (higher is better; each panel uses its own accuracy range), then per-token validation loss and smoothed training loss (lower is better). At this scale LastKV runs the Reuse KV construction against the full-cache Transformer.

4.4 Ways to Obtain the Last-Layer KV

Deciding that the last layer owns the retained cache does not yet say what is stored. Figure 3 compares the three constructions we consider, all of which keep exactly one tensor per completed chunk — so the $O(TCd)$ accounting of Table 1 is identical for all three — and differ only in how a reading layer turns that tensor into keys and values. Writing $\mathbf{H}_t^{(L)}$ for the completed chunk's last-layer feature and $\phi^{(\ell)}$ for the map layer ℓ applies to the retained entry:

$$\begin{aligned} \textit{Reuse KV}: &\quad \phi^{(\ell)} \;=\; \text{identity on } \big(\mathbf{K}_t^{(L)}, \mathbf{V}_t^{(L)}\big), \\ \textit{Shared KV}: &\quad \phi^{(\ell)}\big(\mathbf{H}_t^{(L)}\big) = \big(\mathbf{H}_t^{(L)} W_K^{\mathrm{mem}},\; \mathbf{H}_t^{(L)} W_V^{\mathrm{mem}}\big), \\ \textit{Per-layer KV}: &\quad \phi^{(\ell)}\big(\mathbf{H}_t^{(L)}\big) = \big(\mathbf{H}_t^{(L)} W_K^{(\ell)},\; \mathbf{H}_t^{(L)} W_V^{(\ell)}\big). \end{aligned} \tag{19}$$

Reuse KV retains the last layer's attention keys and values as they were already computed, so every layer reads the identical entry and the construction adds no parameters and no arithmetic. *Shared KV* instead retains the last-layer feature and passes it through one additional projection pair that all layers share, adding $2d^2$ parameters dedicated to memory. *Per-layer KV* retains the same feature but lets each layer apply its *own* existing key and value projections to it, adding no parameters but requiring L separate projections of the retained history at read time. The three therefore trade the same memory budget against increasing amounts of per-layer specialization: one entry read verbatim, one entry read through a shared learned lens, one entry read through L different lenses.

Two observations follow from Figure 3. On validation loss the three are almost indistinguishable — 2.5373, 2.5366, and 2.5375 nats for Reuse, Shared, and Per-layer against 2.5423 for the full-cache Transformer (Table 14) — so all of them recover essentially the same language-modeling quality, and the extra projections buy under 10^{-3} nats. On retrieval the constructions do separate, but mainly at short context: averaged over the 13 RULER tasks at 760M, Per-layer KV is the strongest at 4K (60.6% against 50.6% for the full cache and 48.8% for Reuse) and keeps an edge at 8K and 16K, while by 32K all three collapse into a narrow band (30.4–31.6%) that no construction escapes. Specialization helps a reading layer interpret the shared entry when there is little history to search, and stops mattering once the history is long.

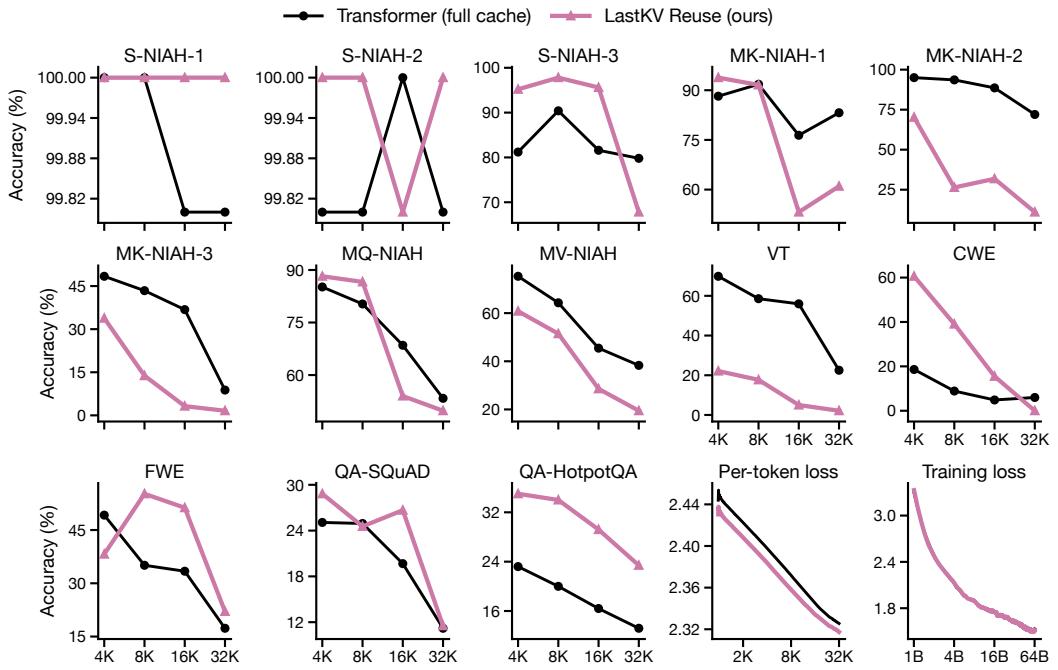


Figure 5: 3B results with gated attention, laid out as in Figure 3: per-task RULER accuracy at context lengths 4K–32K in the first 13 panels (higher is better; each panel uses its own accuracy range), then per-token validation loss and smoothed training loss (lower is better). At this scale LastKV runs the Reuse KV construction against the full-cache Transformer; both models use step-30720 checkpoints.

We use *Reuse KV* for every LastKV model in this paper. It matches the alternatives on loss, it is within the same band at long context where the design is actually aimed, and it is the only one of the three that is free: no memory parameters, no extra projections at read time, and no choice of where to insert them. The video-generation counterpart in Section 6 (Tables 4 and 6) shows the same ordering, which is why the same default carries across modalities.

4.5 Which Layer Should Be Kept?

We sweep which layer supplies the retained cache for the 760M LastKV Reuse-KV model. Figure 6 varies the source index, where source 23 — the last layer — is the main configuration. Deeper sources are consistently stronger, especially at long context: the lowest indices lose S-NIAH retrieval entirely beyond 8K, and the per-token loss panel separates the same way, with source 0 the only curve that visibly flattens. This is the empirical answer to the question left open in Section 3: among all legal source layers, the last one is at or near the best, and it is also the one that costs nothing to produce.

4.6 Analysis for Recurrent KV

How far back the shared cache reaches. Figure 7 sweeps vgap, which controls how much of the recurrent history is condensed into the shared cache; vgap 95 is the setting used everywhere else in the paper. Accuracy is broadly flat across the middle of the range and degrades at the extremes, so the design is not delicately tuned — but the flatness is task-dependent, and the aggregation tasks are the most sensitive.

Chunk size as cache-write frequency. The chunk size controls how frequently the model writes last-layer KV memory: a smaller chunk triggers more frequent writes, while a larger chunk relies more on within-chunk attention. Table 13 reports per-token validation loss as the chunk size varies at fixed context length. Smaller chunks reach lower loss, and they also retain less cache — the nearest-chunk term of Eq. (6) shrinks with C — at the cost of more frequent chunk boundaries during training.

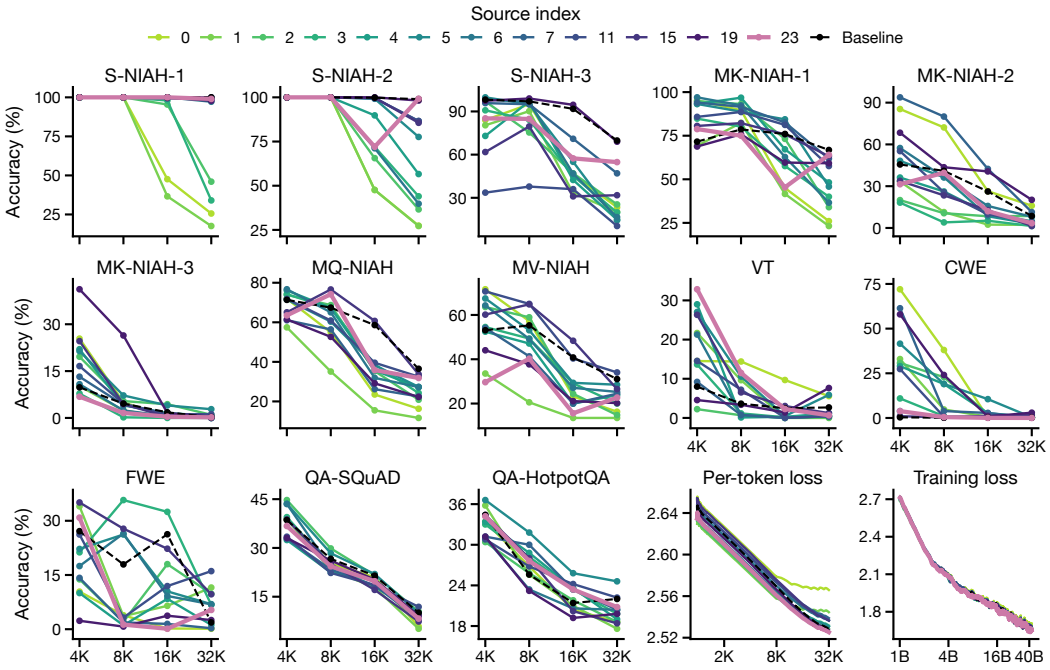


Figure 6: Source-index sweep for the 760M LastKV Reuse-KV model (akv254): per-task RULER accuracy at context lengths 4K–32K, one line per source index, colored on a yellow-green-to-blue gradient with larger indices darker; source 23 — the last layer, i.e. the main LastKV configuration — is drawn in pink, and the black dashed line is the full-cache Transformer baseline. Each panel uses its own accuracy range. The last two panels (lower is better) show per-token validation loss over a log-scale token index (time-weighted EMA smoothing 0.99) and smoothed training loss over log-scale training tokens; the source-15 training run is unavailable.

4.7 KV Share from the Top or from the Bottom?

The source index also settles the direction question raised in the section opener. A small source index is downward sharing — upper layers read a cache produced by a low layer, the direction prior cross-layer designs are forced into — while the last layer is upward sharing, every layer reading the deepest cache, which only chunked recurrence makes possible. Figure 8 isolates this axis in the plain gated-attention Transformer, where a single fixed layer's KV is shared across all layers exactly as in YOCO-style single-source designs, and overlays LastKV for direct comparison. Reading upward is the better direction: the lowest source indices are the weakest curves in almost every panel, quality improves monotonically as the source moves up the stack, and LastKV sits at or above the best single-source setting on the QA and aggregation tasks while matching it elsewhere.

4.8 KV Cache Nearest-Chunk Blend

Condensing a chunk to one last-layer cache costs the immediately preceding chunk the most, because that is the context a token would otherwise read at full per-layer resolution. Two figures separate this effect. Figure 9 sweeps the source index within the recurrent gated-attention family that does *not* use the blend, so it shows what the source choice buys on its own; Figure 10 then compares the main LastKV model against the same last-layer-reuse variant trained without the blend. The clearest evidence is in the loss: without the blend, per-token loss spikes at every 4K chunk boundary — $+0.118$ nats averaged over the 64 tokens after a boundary, versus $+0.002$ with it — reproducing the artifact of Figure 2, while the two training curves are indistinguishable. On RULER the trade is not one-sided: the blend helps on the single-needle and QA tasks at long context and hurts on CWE and the multi-key tasks, so its role is boundary continuity rather than uniform improvement. Table 15 gives the numeric version.

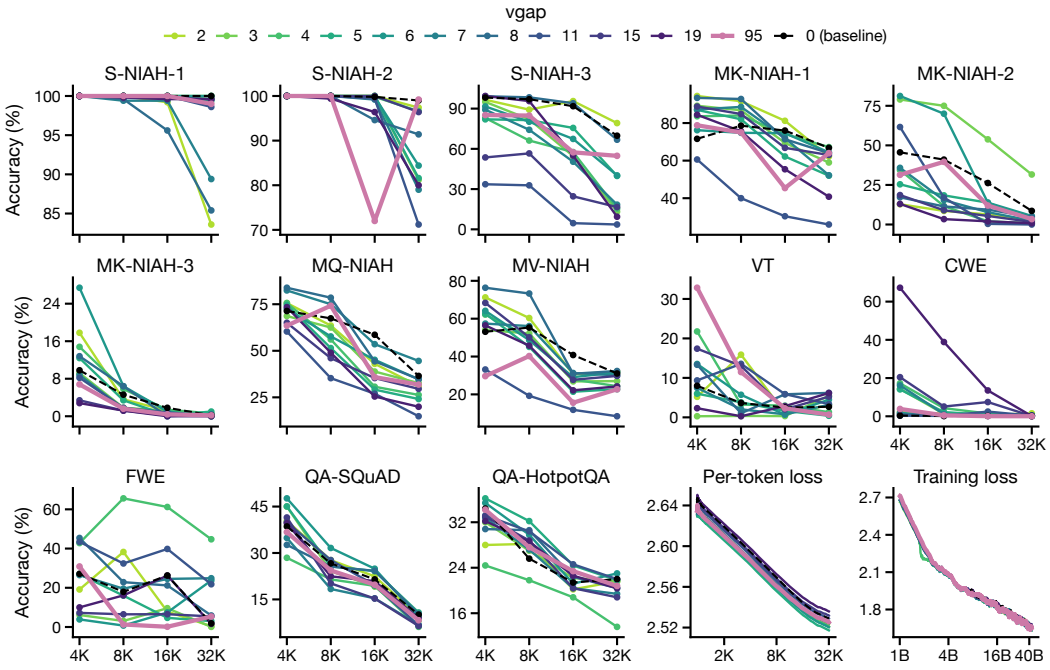


Figure 7: vgap sweep for the 760M LastKV Reuse-KV model (akv254): per-task RULER accuracy at context lengths 4K–32K, one line per vgap value, colored on a yellow-green-to-blue gradient with larger values darker, with the full-cache Transformer baseline as a black dashed line; vgap 95, the main configuration used elsewhere in the paper, is drawn in pink, and vgap 1 is omitted. The last two panels (lower is better) show per-token validation loss over a log-scale token index (time-weighted EMA smoothing 0.99) and smoothed training loss over log-scale training tokens.

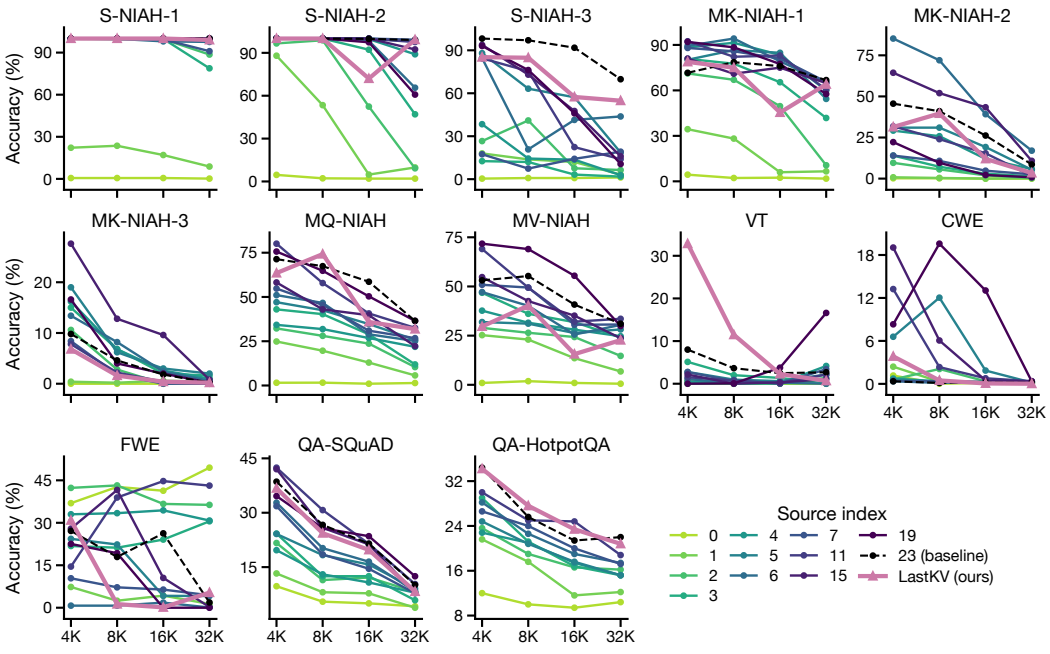


Figure 8: Source-index sweep for the 760M gated-attention Transformer, laid out as in Figure 6: each source index shares one fixed layer's KV across all layers, i.e. a YOCO-style single-source design. The pink line overlays LastKV (Reuse KV, per-chunk last-layer write) and the black dashed line is the full-cache baseline. The source-15 and source-19 evaluations are partial exports and are absent from the QA-HotpotQA panel.

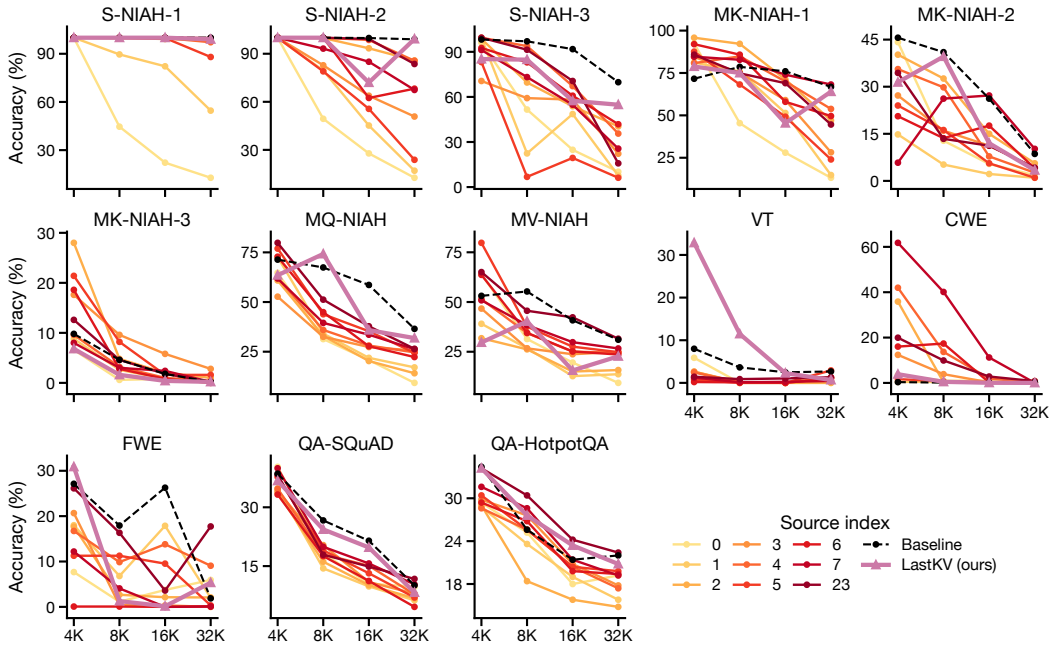


Figure 9: Source-index sweep for the 760M recurrent gated-attention Transformer, laid out as in Figure 6 but drawn on a separate yellow-to-red ramp because these runs are a different model family: at a small source index every layer reads downward from a low layer's cache, while at source 23 all layers read upward from the last layer. The pink line overlays LastKV (Reuse KV, akv254) and the black dashed line is the full-cache Transformer baseline.

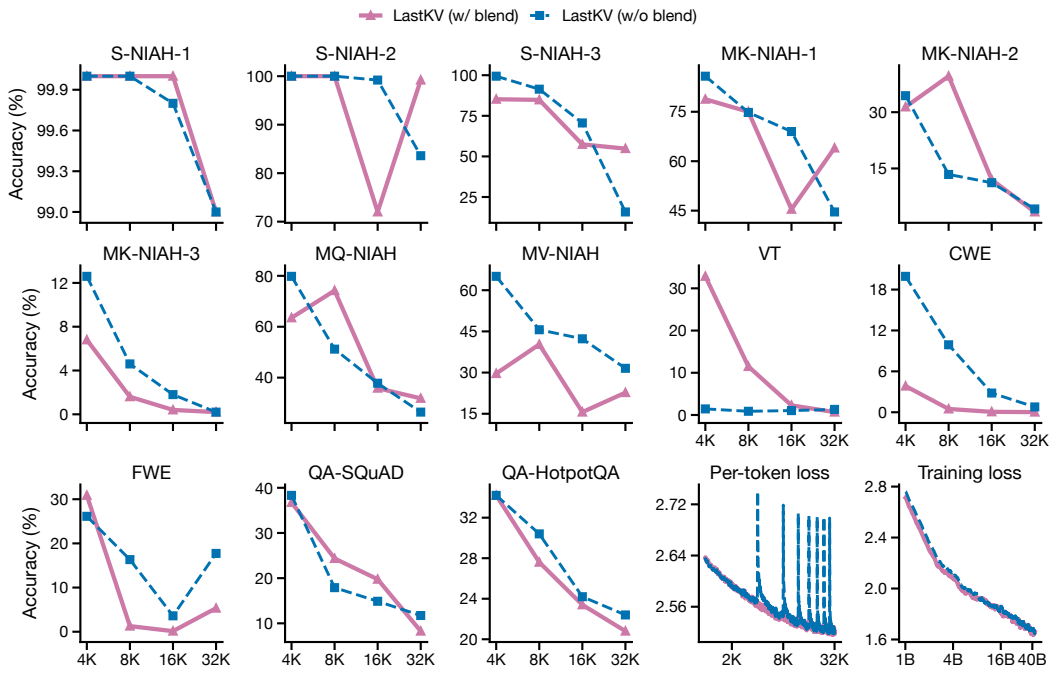


Figure 10: Blend analysis: per-task RULER accuracy at context lengths 4K–32K for the 760M LastKV Reuse-KV model with the nearest-chunk blend (the main configuration) and for the last-layer-reuse variant trained without it. Each panel uses its own accuracy range. The last two panels (lower is better) show per-token validation loss over a log-scale token index and smoothed training loss over log-scale training tokens. Without the blend, per-token loss spikes at every 4K chunk boundary — the artifact of Figure 2 — while the blend removes it; since those spikes are only tens of tokens wide, this panel uses a 32-token centered moving average rather than the EMA smoothing of Figures 6 and 7.

5 LLM Continued Pretraining

Nearly every model in deployment was pretrained as an ordinary full-cache Transformer. For those models to enjoy the single-KV-cache benefit, the architecture must be something an existing checkpoint can be converted *into*, not only something trained from scratch as in Section 4. This section tests exactly that: converting a pretrained Transformer into the LastKV architecture by continued pretraining. The conversion is architecturally cheap — the block is unchanged, the only new parameters are the $L \cdot H$ blend scalars of Section 4.1, and every weight of the original model transfers unmodified — so what has to be learned is not a new module but a new habit: layers that were trained to read their own per-layer history must learn to read a shared last-layer cache instead.

Motivation. The depth argument of Section 4 cuts hardest here. Depth — real depth in parameters, or virtual depth from looping a block at test time [Dehghani et al., 2019, Geiping et al., 2025] — is where capability comes from, and every added layer, real or looped, adds another per-layer KV cache at inference. A deep model therefore *must* share KV across layers if its serving cost is to stay bounded; the question is only how far sharing can be pushed. In the write-during-forward regime the answer is: not to one layer. A YOCO-style design restricted to a single cache can only share the *first* layer's KV, since that is the only cache every other layer can wait for — and we trained this configuration: it does not produce a working model, consistent with the collapse of the lowest source indices in Figure 8. Moving the source to the midpoint yields a usable model, but every layer up to the source still keeps its own cache during generation, so the retained depth is only halved. Chunked recurrence is, to our knowledge, the only route to a model that works with genuinely one layer of KV: deferring the write to the chunk boundary lets all layers read the last layer's cache (Section 3). Recurrence may also bring benefits of its own — the from-scratch runs of Section 4 already show better validation loss per token, and we hypothesize the recurrent state helps state-tracking workloads, which is what agentic tasks are, and reasoning — and it admits a second reading: the chunk loop converts sequence *length* into loop *depth* at decoding time, a form of virtual depth that grows with the context rather than with the parameter count.

Training Throughput. We take this conversion to deployment scale on Marin 8B [The Marin Community, 2025], an openly released 8B full-cache Transformer, and measure the training throughput of its LastKV form at 8K context (Table 2). With an optimized chunk-loop implementation the compact cache costs little throughput at the larger chunks: forward+backward throughput reaches 95.6% of the full-cache Transformer at $C = 1024$ and 88.0% at $C = 512$, and the forward pass is within 0.4% of the baseline at $C = 1024$; only at small chunks does it fall off, to 52.0% at $C = 128$ where chunk boundaries dominate. The retained-cache reduction therefore need not trade against training throughput at this scale.

Experimental Setup. We start from the Phoenix checkpoint of Marin 8B and continue pretraining both architectures on the same 2T-token mixture — half web text and half code and mathematics, curated from Nemotron-CC [Su et al., 2024a] — with the same schedule and data order. Training runs on 448 H200 GPUs with 8 sequences of 8K tokens per GPU, a global batch of 3,584 sequences ($\approx$29.4M tokens per optimizer step), peak learning rate 1.7×10^{-3}, and a 1,024-step warmup. The LastKV model (Reuse KV, chunk size 1024, matching the best-throughput setting of Table 2) and the unconverted full-cache baseline differ only in the retained-cache scheme. We evaluate full checkpoints on a 512-step cadence across thirteen benchmarks spanning knowledge (MMLU, MMLU-Pro, C-Eval, CMMLU, TriviaQA), reasoning (BBH, ARC-C, GPQA-Diamond, HellaSwag, WinoGrande, TruthfulQA), and mathematics (GSM8K, MATH-500), reporting both the raw checkpoint and an exponential moving average (EMA) over the last 200 optimizer steps. LastKV checkpoints are served by a fused chunk-loop vLLM kernel that agrees with HuggingFace scoring to within 0.2σ.

Scope. The goal of this section is deliberately modest: to establish that a single-layer-KV model at 8B scale *works* — that conversion does not stall midtraining and the converted model scores like a normal model of its size. We do not claim superiority over the full-cache Transformer at this scale, and we do not have the compute to train two 8B models to the full 2T budget side by side; the two-architecture comparison is therefore confined to the early part of the budget (the first $\sim$200B tokens, about 6.8k optimizer steps), where the claim is parity, not advantage. Matching token budgets is itself only the academically fair comparison: under an *inference-cost*-fair comparison, the HBM

Table 2: Training throughput of LastKV converted from Marin 8B, at 8K context, per GPU and relative to the full-cache Transformer. Measured on H200 GPUs with batch size 8 per GPU and full activation checkpointing. Larger chunks recover almost all of the baseline's throughput.

	Forward		Forward + Backward	
Method	Tok/s/GPU	rel.	Tok/s/GPU	rel.
Transformer (full cache)	27,759	100.0%	8,181	100.0%
LastKV, $C = 128$	18,183	65.5%	4,253	52.0%
LastKV, $C = 256$	22,791	82.1%	6,032	73.7%
LastKV, $C = 512$	26,021	93.7%	7,201	88.0%
LastKV, $C = 1024$	27,658	99.6%	7,818	**95.6%**

Table 3: Continued pretraining from the Marin 8B Phoenix checkpoint on the same 2T-token Nemotron-derived mixture: accuracy (%) at the matched optimizer step 4,096 (≈120B tokens; raw checkpoint scores on both sides, no EMA), against the shared starting checkpoint. LastKV converts the checkpoint to Reuse KV with chunk size 1024; the baseline continues unconverted; deltas are computed before rounding. Snapshot of an ongoing run. The right block quotes openly released dense base models of the same size class as reported in the Qwen2.5 technical report [Qwen Team, 2024] under its own evaluation protocol (different few-shot settings; ‡GPQA there is the full set rather than Diamond, and MATH is the full test set rather than MATH-500), so those columns are context rather than controlled comparisons.

					Open dense base models (reported)			
Benchmark	Start ckpt	Transformer	LastKV	Δ	Mistral-7B	Llama3-8B	Gemma2-9B	Qwen2.5-7B
MMLU	58.0	67.2	67.3	+0.0	64.2	66.6	71.3	74.2
BBH	39.4	65.5	62.6	−2.9	56.1	57.7	68.2	70.4
HellaSwag	81.2	79.9	79.8	−0.1	83.3	82.1	81.9	80.2
WinoGrande	76.4	76.2	76.0	−0.2	78.4	77.4	79.5	75.9
ARC-C	59.0	62.6	61.8	−0.9	60.0	59.3	68.2	63.7
TruthfulQA	45.9	51.2	49.6	−1.6	42.2	44.0	45.3	56.4
GPQA-D	25.8	35.9	35.9	+0.0	24.7‡	25.8‡	32.8‡	36.4‡
TriviaQA	62.6	58.9	58.6	−0.3	–	–	–	–
C-Eval	40.2	49.9	50.0	+0.1	–	–	–	–
CMMLU	39.2	49.7	49.5	−0.2	–	–	–	–
GSM8K	20.6	65.0	67.6	+2.7	36.2	55.3	70.7	85.4
MATH-500	7.0	16.2	12.8	−3.4	10.2‡	20.5‡	37.7‡	49.8‡

that LastKV frees could be spent on compute — added depth, real or looped — and that is the regime the design is aimed at; we leave it to future work.

Results. Table 3 compares the two architectures at the same optimizer step on the same data — step 4,096, i.e. ≈120B tokens into the schedule, inside the early-budget window above — using raw checkpoint scores on both sides. Both climb steeply from the shared starting checkpoint — GSM8K from 20.6 to the mid-60s, BBH from 39.4 to the low 60s — so conversion does not slow midtraining down, and at the matched checkpoint the converted model is within about one point of the full-cache baseline on eight of twelve benchmarks (mean gap -0.6). The largest single-benchmark gaps go both ways — GSM8K $+2.7$ in LastKV's favor, MATH-500 -3.4 and BBH -2.9 against — which is the scatter single-checkpoint scores show from step to step at this cadence. A model that retains one layer of KV out of 32 is, at this budget, scoring like the model it was converted from. The reference columns of Table 3 place both runs among openly released dense base models of the same size class [Qwen Team, 2024, Gemma Team, 2024, Dubey et al., 2024, Jiang et al., 2023]: read as context only, since evaluation protocols differ, the converted model already scores in the same band as these $\leq$10B bases on knowledge and reasoning benchmarks while its math scores continue to climb with the ongoing schedule.

6 Autoregressive Video Generation

We next instantiate LastKV in autoregressive video generation. A full-attention video diffusion model that is unrolled autoregressively keeps a per-layer KV cache for every past latent frame; LastKV instead keeps a single shared last-layer cache, an L-fold reduction in the memory that grows with clip length. We fine-tune a pretrained text-to-video diffusion model into such a causal generator and ask whether this compact history preserves generation quality under both teacher forcing and self-forcing distillation.

6.1 Method

For autoregressive video generation, each chunk contains latent video tokens from VAE encoding and patchification. Chunks are causal in time: chunk $t{+}1$ may read states from chunks $\leq t$, but never future chunks. We use the same last-layer KV transition as language modeling, together with block-causal local attention inside each video chunk. To avoid propagating denoising noise, the memory write is computed from a clean forward pass over chunk t. This pass reuses the same denoising network, but takes clean frame latents as input; see Self-Forcing [Huang et al., 2025] for the corresponding clean-context construction. Let $\tilde{\mathbf{H}}_t^{(\ell)}$, $\tilde{\mathbf{K}}_t^{(L)}$, and $\tilde{\mathbf{V}}_t^{(L)}$ denote the clean hidden states and last-layer KV tensors. The clean retained cache is

$$\mathbf{S}_t^{\text{clean}} = \big(\tilde{\mathbf{K}}_t^{(L)},\ \tilde{\mathbf{V}}_t^{(L)}\big). \tag{20}$$

When denoising chunk $t{+}1$, each layer attends to clean retained caches $\{\mathbf{S}_\tau^{\text{clean}}\}_{\tau\leq t}$ via Eqs. (13)–(14). At inference time, every generated chunk is reprocessed as clean context before its memory is written. The clean pass adds one context forward whenever a video chunk is committed to memory, but it adds no separate memory model; only the last-layer KV cache, with size $O(Cd)$ per chunk, is stored for older video history.

6.2 Experiment Setup

Teacher-Forcing. We fine-tune Wan 2.1 [Wan Team, 2025] T2V 1.3B on a text-video dataset, converting it to an autoregressive generator with causal last-layer KV reuse: previous latent-frame chunks write last-layer KV caches, and each layer reads them through ordinary block-causal attention over the concatenated context, Eqs. (13)–(14), with the nearest-chunk refinement of Section 3.4. We keep Wan's causal video VAE and $2 \times 2 \times 1$ patchification; 81-frame, 16-FPS clips at 480×832 yield 21 latent frames of size 60×104 ($30 \times 52 = 1560$ tokens per latent frame), processed as chunks of three latent frames (4680 tokens per chunk). LastKV uses the *two-forward* training scheme (a clean context pass builds the retained caches, then the denoising pass reads them; Appendix A) together with the first-token cache exemption of Appendix A. The full-cache teacher-forcing baseline uses a single-pass FlexAttention implementation, which interleaves clean and noisy chunks in one sequence under a compiled block mask (both implementations are detailed in Appendix A). Training runs for 100K steps with batch size 32, learning rate 1×10^{-5}, AdamW betas $(0.9, 0.98)$, weight decay 0.01, cosine decay with 1K warmup, gradient clipping 1.0, timestep shift 5.0, text dropout 0.1. Validation averages denoising loss over timesteps $\{50, 150, 250, 400, 600, 800\}$ every 50 iterations. Each experiment uses 32 A100 80G GPUs. Both methods fine-tune the same Wan 2.1 1.3B checkpoint with the identical recipe; LastKV additionally recomputes a clean context pass when a chunk is committed to memory (one extra context forward per committed chunk), while the baseline retains the full per-layer cache of all previous latent frames.

Self-Forcing. Teacher-forcing writes memory from groundtruth video latents during training but from generated latents at inference, creating exposure bias. To test robustness under this mismatch, we follow the official Self-Forcing setup [Huang et al., 2025]: we distill a 50-step Wan 2.1 14B T2V teacher into a 4-step Wan 1.3B T2V student with one-frame chunks while computing last-layer KV cache from self-generated latents during training. We first run ODE initialization on 16K teacher-generated trajectories for 8K iterations with batch size 128, then train on VidProM captions [Wang and Yang, 2024] for 1K iterations with batch size 64 using the 4-step denoising schedule [1000, 750, 500, 250]. We evaluate 5-second generations on the official VBench [Huang et al., 2024]. Each experiment uses 64 A100 80G GPUs.

Table 4: VBench per-dimension scores for the 1.3B 480p teacher-forcing text-to-video models under the official VBench protocol (4,730 videos; %; higher is better). Left to right: the two full-cache Transformer baselines (single-pass FlexAttention; two-forward), then LastKV variants with the first-token cache exemption (our final model uses Reuse KV) and without it. Bold marks the best value in each row.

	Transformer (full cache)		LastKV, first-token exemption			LastKV, no exemption		
Dimension	FlexAttn	Two-fwd	**Reuse (ours)**	Per-layer	Shared	Reuse	Per-layer	Shared
Quality (aggregate)	**83.33**	83.15	83.11	83.12	82.98	83.08	82.80	82.86
Semantic (aggregate)	77.14	77.07	76.75	77.02	77.35	76.89	76.76	**78.59**
Total	**82.09**	81.94	81.84	81.90	81.85	81.84	81.60	82.01
Subject consistency	93.27	93.83	93.33	**93.91**	93.60	92.84	92.31	93.05
Background consistency	95.02	**95.67**	94.73	95.31	94.57	94.82	94.22	94.57
Temporal flickering	97.70	97.29	97.80	97.85	97.59	97.58	97.30	**97.89**
Motion smoothness	98.62	98.53	**98.68**	98.65	98.66	98.56	98.32	98.60
Dynamic degree	81.67	75.00	77.50	74.72	73.06	80.00	**86.11**	74.17
Aesthetic quality	62.01	62.16	61.23	61.90	**62.55**	61.19	61.40	61.61
Imaging quality	63.52	**65.44**	64.78	64.07	65.38	64.85	62.80	65.01
Object class	90.46	91.36	**93.80**	92.58	91.61	89.89	90.43	92.80
Multiple objects	74.76	73.69	75.78	74.54	73.26	71.43	74.27	**76.27**
Human action	96.80	96.80	97.20	97.00	96.60	**97.60**	97.40	97.40
Color	86.64	84.86	82.67	84.90	88.24	87.41	84.79	**88.66**
Spatial relationship	67.48	70.02	65.58	68.38	66.60	65.15	66.80	**70.65**
Scene	**55.12**	53.49	50.74	52.51	53.08	54.99	52.06	54.80
Appearance style	21.03	21.66	21.59	21.39	**22.09**	21.70	21.81	21.81
Temporal style	24.11	23.73	**24.29**	24.10	24.08	24.11	24.08	24.22
Overall consistency	25.93	25.78	26.12	25.78	26.16	26.03	25.99	**26.23**

6.3 Main Results

LastKV keeps a single shared last-layer cache for older frame history in place of the per-frame, per-layer cache a full-attention generator retains: for the 1.3B model this is one cache instead of the 30 per-layer caches kept for every past latent frame. Tables 4–6 give the full VBench per-dimension evaluation under two training regimes.

Teacher forcing. At 1.3B (Table 4), LastKV is within 0.25 VBench total points of the single-pass FlexAttention full-cache baseline and within 0.10 of the matched two-forward baseline; at 14B (Table 5) the two are within 0.16 points. This is essentially quality parity at a fraction of the retained cache. The per-dimension breakdown shows the small gap is not uniform: at 1.3B LastKV is best on motion smoothness, object class, and temporal style and trails mainly on scene, color, and spatial relationship; at 14B it leads on dynamic degree, color, and multiple objects and trails on spatial relationship, imaging quality, and subject consistency.

Self forcing. Under self-forcing distillation, which writes memory from the model's own generations and is the harder regime, LastKV trails the full-cache two-forward baseline on the aggregate VBench total at every distillation checkpoint, by 0.4–0.6 points (83.18 vs. 83.69 at the final $1{,}000$-step checkpoint; Table 6). The per-dimension breakdown shows this aggregate gap is almost entirely a dynamic-degree effect: the full-cache baseline scores 66.1 on dynamic degree against 38.9 for LastKV, while on the consistency, aesthetic, and semantic dimensions LastKV is competitive or better ($+1.7$ subject consistency, $+1.3$ background consistency, $+2.7$ color, $+0.4$ semantic aggregate). Last-layer reuse thus trades motion magnitude for temporal stability under distillation rather than losing quality broadly. Table 6 also isolates LastKV's two design choices (columns 2–4): the first-token sink exemption helps consistently ($+0.6$ total points, Reuse+sink vs. Reuse-no-sink), and Reuse versus Shared KV is within noise (Shared 0.17 ahead on the total). Because Reuse KV takes the last layer's attention KV directly with no added parameters or FLOPs while Shared KV needs an extra projection for a gain inside this noise band, we keep Reuse KV with the sink exemption as the final video model.

Table 5: VBench per-dimension scores for the 14B 480p teacher-forcing text-to-video models (official VBench protocol, 4,730 videos; %; higher is better). The full-cache baseline uses the single-pass FlexAttention implementation. Bold marks the best value in each row.

Dimension	Transformer (FlexAttention)	LastKV (Reuse KV)
Quality (aggregate)	82.17	**82.64**
Semantic (aggregate)	**72.74**	71.69
Total	80.29	**80.45**
Subject consistency	**93.40**	91.28
Background consistency	**95.29**	94.73
Temporal flickering	96.57	**97.70**
Motion smoothness	98.44	**98.48**
Dynamic degree	68.06	**84.17**
Aesthetic quality	**61.05**	59.34
Imaging quality	**66.94**	63.66
Object class	**86.22**	85.43
Multiple objects	66.33	**69.37**
Human action	**95.00**	92.60
Color	80.30	**85.82**
Spatial relationship	**68.27**	60.94
Scene	**48.90**	47.22
Appearance style	20.30	**20.62**
Temporal style	**22.19**	20.82
Overall consistency	**24.41**	23.41

6.4 Analysis

How to Build the Retained KV Cache? The KV-construction choice — *Reuse*, *Shared*, or *Per-layer* — is ablated directly in the per-dimension tables (Tables 4 and 6). Across teacher forcing and self forcing, *Reuse KV* stays within 0.2 VBench total points of the best construction in every case, and it reuses the last layer's attention KV at no added parameters or FLOPs, so LastKV adopts it, matching the language-model ablation of Section 4.

First-Token Sink. The retained cache exempts the very first token: each layer keeps its own key–value pair for that token rather than condensing it to the last layer's, both in the shared cache and inside the nearest-chunk blend (Appendix A). This follows the attention-sink behavior of initial tokens [Xiao et al., 2024] — collapsing the first token to a single shared entry removes each layer's sink. The ablation confirms it is worth keeping: removing the exemption costs up to 0.6 VBench total points under self forcing (Reuse+sink vs. Reuse-no-sink in Table 6), and Table 4 gives the teacher-forcing per-dimension version of the same ablation.

Table 6: Self-forcing VBench per-dimension scores at 1,000 distillation steps (1.3B student; official protocol, 4,730 videos; %; higher is better). Columns: the full-cache two-forward baseline, then LastKV with the first-token sink exemption (Reuse KV, our final model) and the two no-sink variants (Reuse and Shared KV). Bold marks the best value in each row. The aggregate-total gap is concentrated in dynamic degree; LastKV leads on most consistency and semantic dimensions.

	Full cache	LastKV		
Dimension	Two-forward	**Reuse+sink (ours)**	Reuse, no sink	Shared, no sink
Quality (aggregate)	**84.45**	83.73	83.09	83.25
Semantic (aggregate)	80.64	**80.99**	80.50	80.71
Total	**83.69**	83.18	82.57	82.74
Subject consistency	95.81	**97.51**	96.36	96.76
Background consistency	95.91	**97.22**	96.26	96.98
Temporal flickering	98.13	99.12	98.74	**99.27**
Motion smoothness	98.23	98.77	98.70	**98.89**
Dynamic degree	**66.11**	38.89	41.67	33.06
Aesthetic quality	67.56	**68.25**	67.00	68.17
Imaging quality	69.01	68.97	68.60	**69.20**
Object class	95.43	95.43	95.35	**96.39**
Multiple objects	**90.34**	87.50	87.23	89.22
Human action	95.60	96.80	**97.40**	95.20
Color	86.86	**89.54**	88.20	85.98
Spatial relationship	81.27	81.85	78.65	**82.11**
Scene	54.08	55.35	**56.06**	55.19
Appearance style	20.15	**20.28**	19.96	20.03
Temporal style	24.22	24.17	24.12	**24.34**
Overall consistency	**26.75**	26.62	26.72	26.72

7 Novel View Synthesis

Finally, we instantiate LastKV in novel view synthesis (NVS), the cleanest test of the memory question. The views of a static scene form an almost independent set, and the retained scene memory is exactly the last-layer cache of the observed views: each observed view writes one shared last-layer cache in place of the eight per-layer caches a full-attention model keeps. Unlike the language and video settings, where the compact cache trades a little quality for the memory saving, here it *improves* reconstruction over full attention.

7.1 Method

We formulate novel view synthesis as *streaming reconstruction autoregression* over posed views of a static scene, treating each view as one recurrent chunk. Let $\mathcal{X}_m$ denote the visual tokens and camera metadata of view m. Its hidden states and QKV tensors are computed using Eq. (11), with view index m replacing chunk index t. Observed input views contain RGB tokens together with camera pose, while target views contain pose-only query tokens. After an observed input view m has passed through all L layers, it writes its last-layer KV pair into the retained scene cache:

$$\mathbf{S}_m^{\text{view}} = \big(\mathbf{K}_m^{(L)},\ \mathbf{V}_m^{(L)}\big). \tag{21}$$

Subsequent observed or target views attend to context built from previous input-view last-layer caches through Eqs. (13)–(14). Target views are read-only: their generated image tokens are not written back into memory. Thus later targets obtain context from the scene memory written by observed RGB+pose views and from their own pose-only tokens, enabling direct cross-view correspondence without keeping a full per-layer cache for every previous view. Algorithm 3 details the streaming procedure.

7.2 Experiment Setup

We formulate object- and scene-level NVS as streaming autoregressive reconstruction over camera-conditioned views. Object models train on Objaverse [Deitke et al., 2023] and evaluate on the Google Scanned Objects (GSO) [Downs et al., 2022] split from Instant3D [Li et al., 2024a]; scene models train on filtered DL3DV-10K [Ling et al., 2024] and evaluate on its benchmark split. Training uses 12

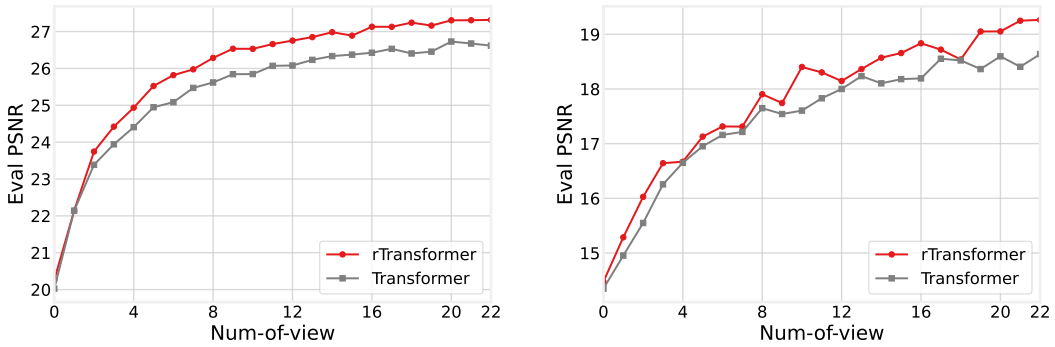
Algorithm 3 LastKV: streaming novel view synthesis (views as chunks)

Require: interleaved view stream $\mathcal{X}_1, \ldots, \mathcal{X}_M$: posed input views (RGB+pose tokens, WRITE) alternating with pose-only target views (READ-ONLY), each tokenized to $\mathbb{R}^{B\times C\times D}$; L blocks with attention projections $W_{QKV}^{(\ell)}$ and a KV-construction map $\phi^{(\ell)}$

$\mathcal{R}_K^{(\ell)}, \mathcal{R}_V^{(\ell)} \leftarrow \emptyset$ for all ℓ ▷ retained scene memory
for $m = 1$ **to** M **do**
  $\mathbf{X} \leftarrow \mathcal{X}_m$
  for $\ell = 1$ **to** L **do**
    $\mathbf{Z}^{(\ell)} \leftarrow \mathrm{RMSNorm}(\mathbf{X})$ ▷ block-ℓ attention input; $\mathbf{Z}^{(L)}$ is the memory feature
    $\mathbf{Q}, \mathbf{K}, \mathbf{V} \leftarrow$ projections of $\mathbf{Z}^{(\ell)}$
    $\mathbf{O} \leftarrow \mathrm{FlashAttn}\big(\mathbf{Q}, [\mathcal{R}_K^{(\ell)}; \mathbf{K}], [\mathcal{R}_V^{(\ell)}; \mathbf{V}]\big)$ ▷ bidirectional; no causal mask
    $\mathbf{X} \leftarrow \mathbf{X} + \mathbf{O}W_O^{(\ell)}$; $\mathbf{X} \leftarrow \mathbf{X} + \mathrm{MLP}^{(\ell)}(\mathrm{RMSNorm}(\mathbf{X}))$
  end for
  render / supervise from $\mathbf{X}$
  if $\mathcal{X}_m$ is a posed input view **then** ▷ targets never write; generated pixels never enter memory
    $\mathbf{Z} \leftarrow \mathrm{RMSNorm}\big(\mathbf{Z}^{(L)}\big)$ ▷ shared last-layer memory feature
    for $\ell = 1$ **to** L **do**
      $(\mathbf{K}_m, \mathbf{V}_m) \leftarrow \phi^{(\ell)}(\mathbf{Z})$ ▷ see KV-construction modes below
      $\mathcal{R}_K^{(\ell)} \leftarrow [\mathcal{R}_K^{(\ell)}; \mathbf{K}_m]$; $\mathcal{R}_V^{(\ell)} \leftarrow [\mathcal{R}_V^{(\ell)}; \mathbf{V}_m]$
    end for
  end if
end for

KV-construction modes (Sec. 3, ablated in Tables 14, 4, and 6): *Reuse KV*: $\phi^{(\ell)}$ returns the last layer's attention $(\mathbf{K}^{(L)}, \mathbf{V}^{(L)})$ directly (no extra compute); *Shared KV*: one shared projection W_{KV}^{mem} applied to $\mathbf{Z}$, identical for all ℓ; *Per-layer KV*: each layer's own $W_{KV}^{(\ell)}$ applied to $\mathbf{Z}$.

views and evaluation uses 24 views, with PSNR as the main metric. Images are resized to 256^2 and tokenized into 8×8 patches, yielding 1024 tokens per view. The sequence alternates posed-image context tokens and pose-only target tokens, giving $2(12-1)\cdot 1024 = 22{,}528$ tokens for training and $2(24-1)\cdot 1024 = 47{,}104$ for evaluation. Only RGB+pose context views write last-layer KV memory; pose-only target views are read-only queries and generated targets are not fed back into memory. We use an 8-layer width-512 backbone. Models train for 20K steps with AdamW, learning rate 4×10^{-4}, weight decay 0.05, 2.5K warmup, gradient clipping at 1.0, and L2 loss. Each experiment uses 8 A100 80G GPUs with a global batch size of 128. The Transformer baseline shares the same backbone and training recipe and differs only in retaining the full per-layer KV cache of every previous view instead of the shared last-layer cache.



(a) Objaverse/GSO. (b) DL3DV.

Figure 11: Novel view synthesis PSNR at each generated view index for the 23 generated views indexed 0 through 22. Left: object-level NVS trained on Objaverse and evaluated on GSO. Right: scene-level NVS trained and evaluated on DL3DV. Higher PSNR indicates more accurate reconstruction; LastKV improves over the Transformer baseline from early generated views.

Table 7: Novel view synthesis PSNR (dB, higher is better) at representative generated-view indices and averaged over all 23 generated views, summarizing Figure 11 in hard numbers. Values are the final-checkpoint per-view evaluation PSNR.

Dataset	Method	View 2	View 6	View 10	View 14	View 18	View 22	Mean
Objaverse/GSO	Transformer	23.39	25.08	25.85	26.34	26.41	26.62	25.38
	LastKV	**23.74**	**25.82**	**26.53**	**26.98**	**27.24**	**27.32**	**25.95**
DL3DV	Transformer	15.55	17.16	17.60	18.10	18.52	18.64	17.46
	LastKV	**16.03**	**17.31**	**18.40**	**18.57**	**18.54**	**19.26**	**17.81**

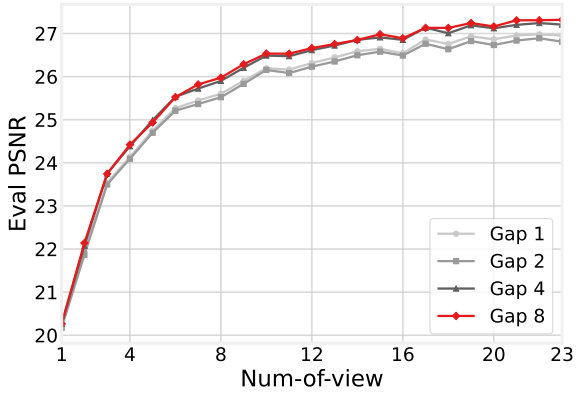


Figure 12: Source-layer ablation for the retained KV cache, measured by NVS PSNR across generated views on Objaverse/GSO in the 24-view evaluation setting. We vary the source gap between the cache injection point and the layer that provides the carried KV; higher is better.

7.3 Main Results

Figure 11 and Table 7 report per-view evaluation PSNR for both settings. LastKV improves the mean over the 23 generated views from 25.38 to 25.95 dB on object-level NVS (Objaverse/GSO) and from 17.46 to 17.81 dB on scene-level NVS (DL3DV); the advantage is present by generated view 2 and grows with view index, reaching $+0.70$ dB at view 22 on Objaverse. That a single shared last-layer cache per view beats the full per-layer cache is the strongest form of the paper's claim: here the compact memory is not merely sufficient but preferable, plausibly because forcing every layer to read one deep summary of each observed view is a useful inductive bias for cross-view correspondence. Two ablations pin down why it works — the last layer is the right source, and the recurrence is what produces the gain.

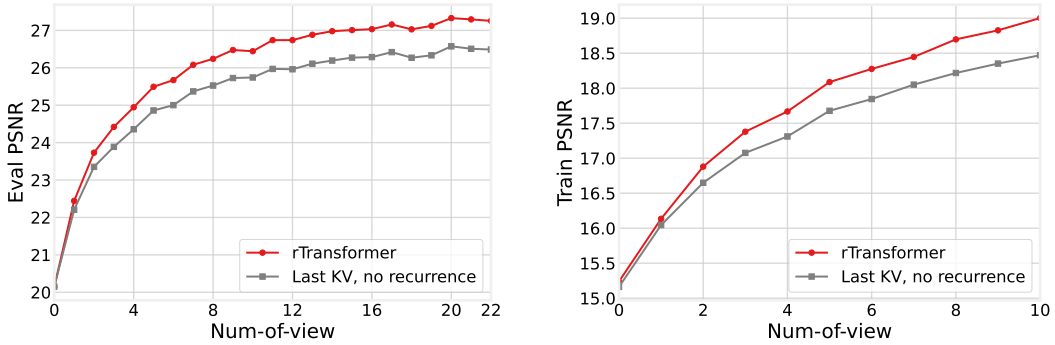
7.4 Analysis

Which Layer Should Be Kept? The design fixes two choices, and NVS lets us test both directly: which layer's cache to retain, and whether to write it recurrently. For the source layer, we carry the retained cache from a layer 1, 2, 4, or 8 blocks above each injection point, where gap 8 is the last layer of the 8-layer backbone. Figure 12 and Table 8 report evaluation PSNR on Objaverse/GSO: the deepest sources are strongest and the last layer is best overall, on both the final-view PSNR (27.32 dB vs. 26.95 at gap 1) and the mean over view counts (25.95 vs. 25.65). This is direct evidence for the last-layer source. The parallel question for language models — which layer should own the cache, and whether layers read it downward or upward — is analyzed in Section 4 (Figures 6–10).

Is the Recurrence Necessary? The source-layer sweep fixes *which* layer supplies the retained cache, but not *how* it is written. We therefore compare LastKV against a variant that keeps the last-layer KV source and removes the chunked recurrence, so that later views no longer read a cache written by completed earlier views. Figure 13 reports both datasets. At generated view 0 the two are indistinguishable — no history has accumulated, so the recurrence has nothing to carry, and the non-recurrent variant is in fact 0.01 dB ahead on Objaverse/GSO — and the gap then opens as soon as views begin to accumulate. On Objaverse/GSO it reaches 0.53 dB by view 3 and settles at roughly

Table 8: Source-layer ablation (NVS, Objaverse/GSO; companion to Fig. 12). Evaluation PSNR of the final generated view (23 views) and the mean over all view counts, as a function of the source gap between the cache injection point and the layer providing the carried KV; gap 8 equals the last layer of the 8-layer backbone. Higher is better.

Source gap	1	2	4	8 (last layer)
Final-view PSNR	26.95	26.81	27.21	**27.32**
Mean PSNR over view counts	25.65	25.57	25.91	**25.95**



(a) Objaverse/GSO, evaluation PSNR.

(b) DL3DV, training-view PSNR.

Figure 13: Removing the chunked recurrence while keeping the last-layer KV source. Higher is better. Left: object-level NVS, per-view evaluation PSNR at the final shared evaluation step (24K). Right: scene-level NVS, per-view PSNR on training views at 20K steps, averaged over the last 20 logged points; this pair of runs did not record per-view evaluation PSNR, so the two panels are not on a common metric and only the within-panel comparison is meaningful.

0.75–0.79 dB from view 11 onward, ending at 0.77 dB at view 22; on DL3DV it grows from 0.08 dB at view 0 to 0.53 dB at view 10. That the advantage is absent exactly where there is no history, and saturates once enough history has been written, is what one expects if the recurrence rather than the last-layer source is responsible: sharing the deepest layer's KV pays off only when a completed view has actually written it into memory for later views to read.

8 Conclusion

LastKV pushes cross-layer KV sharing to its limit for long-range history: each completed chunk keeps only its last-layer cache, and all L layers of every future chunk read that single shared cache. This removes the older per-layer cache dimension while preserving attention-readable memory and chunk-level parallelism; with the nearest-chunk refinement, the practical cache remains $O((T+L)Cd)$ rather than $O(TCLd)$. Because sharing applies only across completed chunks, the within-chunk Transformer and its training procedure are unchanged, in contrast to sharing designs that restructure attention at every position. Across long-context language modeling, novel view synthesis, and autoregressive video generation, controlled studies show that the same last-layer KV principle lowers language-model loss, stays competitive on retrieval, raises NVS PSNR, and preserves aggregate video quality; continued pretraining converts an existing 8B Transformer into LastKV at 95.6% of the baseline's training throughput while matching its benchmark scores. These results suggest that one cross-layer-shared last-layer cache is a compact and effective memory primitive for long-sequence Transformers — and, because the saving grows with depth, one that becomes more valuable as models grow deeper, in parameters or in loops. An end-to-end serving benchmark under a fixed memory budget is the next evaluation step.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Conference on Empirical Methods in Natural Language Processing*, 2023.

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

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*, volume 37, 2024.

Aydar Bulatov, Yury Kuratov, and Mikhail Burtsev. Recurrent memory transformer. *Advances in Neural Information Processing Systems*, 35:11079–11091, 2022.

Ziwen Chen, Hao Tan, Kai Zhang, Sai Bi, Fujun Luan, Yicong Hong, Fuxin Li, and Zexiang Xu. Long-LRM: Long-sequence large reconstruction model for wide-coverage gaussian splats. *arXiv preprint arXiv:2410.12781*, 2024.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734, 2014.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, 2019.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz 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*, pages 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 *IEEE International Conference on Robotics and Automation*, pages 2553–2560, 2022.

Abhimanyu Dubey et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

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.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025.

Gemma Team. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

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

Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J. Fleet. Video diffusion models. In *Advances in Neural Information Processing Systems*, 2022.

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, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024.

Xun Huang, Zhengqi Li, Guande He, Mingyuan Zhou, and Eli Shechtman. Self forcing: Bridging the train-test gap in autoregressive video diffusion. In *Advances in Neural Information Processing Systems*, 2025. Spotlight; arXiv:2506.08009.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, Yaohui Wang, Xinyuan Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. Vbench: Comprehensive benchmark suite for video generative models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

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

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 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 Francois Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 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), 2023.

Jiahao Li, Hao Tan, Kai Zhang, Zexiang Xu, Fujun Luan, Yinghao Xu, Yicong Hong, Kalyan Sunkavalli, Greg Shakhnarovich, and Sai Bi. Instant3D: Fast text-to-3d with sparse-view generation and large reconstruction model. In *International Conference on Learning Representations*, 2024a.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, volume 37, 2024b.

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*, pages 22160–22169, 2024.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. MiniCache: KV cache compression in depth dimension for large language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

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.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *International Conference on Computer Vision*, 2023.

Jackson Petty, Sjoerd van Steenkiste, Ishita Dasgupta, Fei Sha, Dan Garrette, and Tal Linzen. The impact of depth on compositional generalization in transformer language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 7239–7252, 2024.

Zihan Qiu, Zekun Wang, Bo Zheng, Zeyu Huang, Kaiyue Wen, Songlin Yang, Rui Men, Le Yu, Fei Huang, Suozhi Huang, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Gated attention for large language models: Non-linearity, sparsity, and attention-sink-free. *arXiv preprint arXiv:2505.06708*, 2025.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-CC: Transforming Common Crawl into a refined long-horizon pretraining dataset. *arXiv preprint arXiv:2412.02595*, 2024a.

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

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

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*, volume 37, 2024.

The Marin Community. Marin 8B. `https://huggingface.co/marin-community/marin-8b-base`, 2025.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Wan Team. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

Wenhao Wang and Yi Yang. VidProM: A million-scale real prompt-gallery dataset for text-to-video diffusion models. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

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.

You Wu, Haoyi Wu, and Kewei Tu. A systematic study of cross-layer KV sharing for efficient LLM inference. In *Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics*, 2025.

Yuhuai Wu, Markus N. Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *International Conference on Learning Representations*, 2022.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. *arXiv preprint arXiv:2412.06464*, 2024a.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *International Conference on Machine Learning*, pages 56501–56523. PMLR, 2024b.

Table 9: Architecture specifications for all models. The Transformer baseline and LastKV share every dimension, the gated-attention block, tokenizer, and data pipeline at each scale; LastKV adds only the learnable per-layer, per-head nearest-chunk blending logits ($L \cdot H$ scalars: 144 / 576 / 600 for the three LM scales), so all LM comparisons are iso-parameter up to $< 10^{-6}$ relative difference. Parameter counts are computed from the released configs. For NVS and video, one chunk is one view (1024 tokens) and three latent frames (4680 tokens), respectively.

Model	Dim D	Layers L	Heads H	Head dim d_h	MLP dim	Params	RoPE θ	Chunk C	Retained KV/chunk
LM 124M	768	12	12	64	2048	141M	10^6	4096	Cd (vs. $12\,Cd$ full)
LM 760M	1536	24	24	64	4096	835M	10^6	4096	Cd (vs. $24\,Cd$ full)
LM 3B	3072	25	24	128	8192	3.26B	10^6	4096	Cd (vs. $25\,Cd$ full)
NVS	512	8	8	64	1536	28M	–	1024 (1 view)	Cd (vs. $8\,Cd$ full)
Video (Wan 2.1 1.3B)	1536	30	12	128	8960	1.3B	–	4680 (3 latent frames)	Cd (vs. $30\,Cd$ full)

Table 10: Training recipes for all experiments. Each LastKV model and its full-cache Transformer baseline are trained with the identical recipe (same optimizer, schedule, data, tokenizer, steps, and hardware); the LM baseline uses the same gated-attention block, so the retained-cache scheme is the only treatment. Video fine-tuning starts from the same Wan 2.1 1.3B checkpoint for both methods; LastKV additionally recomputes a clean context pass when a chunk is committed to memory (one extra forward per committed chunk, Sec. 3).

Setting	Optim.	(β_1, β_2)	ϵ	LR	Schedule	Warmup	WD	Clip	Global batch	Seq len	Steps	GPUs
LM 124M	AdamW	(0.9, 0.95)	10^{-15}	10^{-3}	cosine $\to 0.1\times$	1024	0.1	1.0	32 ($\sim$1M tok)	32768	10,240	8×A100
LM 760M	AdamW	(0.9, 0.95)	10^{-15}	10^{-3}	cosine $\to 0.1\times$	1024	0.1	1.0	32 ($\sim$1M tok)	32768	40,960	32×A100
LM 3B	AdamW	(0.9, 0.95)	10^{-15}	10^{-3}	cosine $\to 0.1\times$	1024	0.1	1.0	64 ($\sim$2M tok)	32768	30,720	64×A100
NVS	AdamW	(0.9, 0.95)	default	4×10^{-4}	cosine $\to 0$	2500	0.05	1.0	128	22,528	20,000	8×A100
Video (teacher f.)	AdamW	(0.9, 0.98)	default	10^{-5}	cosine	1000	0.01	1.0	32	32,760	100,000	32×A100
Video (self f., ODE init)	AdamW	(0.9, 0.98)	default	10^{-5}	const.	–	0.01	1.0	128	–	8,000	64×A100
Video (self f., distill)	AdamW	(0.9, 0.98)	default	10^{-5}	const.	–	0.01	1.0	64	–	1,000	64×A100

Yifei Yang, Zouying Cao, Qiguang Chen, Libo Qin, Dongjie Yang, Hai Zhao, and Zhi Chen. KVSharer: Efficient inference via layer-wise dissimilar KV cache sharing. *arXiv preprint arXiv:2410.18517*, 2024c.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems*, 2020.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Zayd Muhammad Kawakibi Zuhri, Muhammad Farid Adilazuarda, Ayu Purwarianti, and Alham Fikri Aji. MLKV: Multi-layer key-value heads for memory efficient transformer decoding. In *Findings of the Association for Computational Linguistics: NAACL 2025*, 2025.

A Implementation Details

Algorithm 1 gives the general chunk loop and its $O((T+L)Cd)$ retained state, Algorithm 2 the chunk-parallel training forward for language modeling, and Algorithm 3 the streaming novel-view-synthesis variant. Table 9 specifies every model, and Table 10 the full training recipes. Cache appends are implemented as a custom autograd function writing into preallocated storage, so gradients flow into retained last-layer keys and values from all future chunks without detaching.

Nearest-chunk blending coefficient. The blending coefficient of Eq. (15) is implemented as a learnable per-layer, per-head logit $a^{(\ell)} \in \mathbb{R}^H$ passed through a sigmoid, initialized to 0 so that

Table 11: Full RULER results for the 760M models *with* gated attention (accuracy, %; higher is better): 13 tasks at context lengths 4K–32K, comparing the full-cache Transformer with the three LastKV KV-construction modes (our language models use Reuse KV). Bold marks the best method for each task–length cell. Mean rows average the 13 tasks. Retrieval-heavy multi-key tasks favor the full cache at long contexts, while LastKV variants match or exceed the baseline on single-needle, QA, and aggregation tasks; Per-layer KV is strongest at 4K. Table 12 repeats the comparison without gated attention.

	Transformer (full cache)				LastKV Reuse (ours)				LastKV Per-layer				LastKV Shared			
Task	4K	8K	16K	32K	4K	8K	16K	32K	4K	8K	16K	32K	4K	8K	16K	32K
S-NIAH-1	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	99.0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
S-NIAH-2	**100.0**	**100.0**	99.8	99.0	**100.0**	**100.0**	72.0	**99.2**	**100.0**	78.4	95.4	65.2	**100.0**	93.6	**100.0**	91.0
S-NIAH-3	98.2	97.0	**91.8**	**69.8**	85.2	84.8	57.4	54.8	83.4	4.6	33.4	37.0	**99.4**	**98.4**	81.0	32.6
MK-NIAH-1	71.6	78.6	76.0	**66.8**	78.8	75.0	45.4	64.0	**89.2**	**86.2**	**79.0**	62.4	71.2	61.6	48.4	61.2
MK-NIAH-2	45.6	41.0	**26.2**	**8.6**	31.4	39.6	12.0	3.4	**88.0**	**71.4**	12.6	6.0	36.2	21.6	15.8	6.8
MK-NIAH-3	9.8	**4.6**	**1.8**	0.2	6.8	1.6	0.4	0.2	**20.0**	4.0	1.2	0.2	5.4	0.8	1.2	**0.4**
MQ-NIAH	71.4	67.4	**58.6**	**36.5**	63.5	**74.2**	35.9	31.9	**75.3**	69.4	45.5	32.5	53.2	43.3	24.1	26.9
MV-NIAH	53.1	55.3	40.8	**31.1**	29.6	40.2	15.6	22.7	**63.7**	**68.5**	**42.4**	30.7	32.3	26.2	22.8	26.9
VT	8.0	3.6	**2.5**	2.7	32.8	**11.5**	2.3	0.7	5.5	2.1	1.2	**9.6**	**38.0**	2.3	1.1	4.8
CWE	0.4	0.1	0.2	0.1	3.8	0.5	0.1	0.0	**44.0**	**19.5**	**3.8**	**3.7**	27.2	3.1	0.5	0.1
FWE	27.1	17.9	26.3	1.9	30.9	1.3	0.1	5.3	**47.5**	**50.7**	**30.5**	**20.8**	32.9	8.4	7.9	14.9
QA-SQuAD	**38.6**	26.7	21.5	**10.1**	36.8	24.3	19.8	8.3	38.1	**27.3**	**22.5**	7.7	38.4	26.2	19.6	9.6
QA-HotpotQA	**34.4**	25.6	21.4	**22.0**	34.2	27.6	23.4	20.8	32.8	29.4	**27.0**	20.6	33.6	**33.0**	22.8	20.0
Mean (13 tasks)	50.6	**47.5**	**43.6**	**34.5**	48.8	44.7	29.6	31.6	**60.6**	47.0	38.0	30.5	51.4	39.9	34.2	30.4

blending starts at 0.5; the logits are excluded from weight decay. This adds $L \cdot H$ scalar parameters (576 at 760M).

Teacher-forcing implementations: FlexAttention vs. two-forward. Teacher-forcing video training must let noisy chunks attend to clean context. We use two implementations. The *FlexAttention* baseline concatenates clean and noisy chunks into one interleaved sequence and runs a single forward pass under a compiled block mask in which each noisy chunk attends to all preceding clean chunks and to itself, with full per-layer caches. The *two-forward* scheme instead runs the same denoising network twice per step: a clean context pass over the clean latents at timestep 0, processed chunk by chunk, whose only role is to write the retained caches (for LastKV: the shared last-layer cache plus the per-layer nearest-chunk history; for the two-forward baseline: full per-layer caches), followed by a denoising pass over the noisy chunks at sampled timesteps that reads the cached context and receives the loss. LastKV requires the two-forward scheme because its cache write is defined on completed clean chunks; Table 4 therefore reports the full-cache baseline under both implementations, separating the effect of the training scheme from the effect of the retained cache.

First-token cache exemption (video). In the video realization, the retained-cache entry of the very first token keeps each layer's own key–value pair rather than the last layer's, both when the shared cache is written and inside the nearest-chunk blend. This is motivated by the attention-sink behavior of initial tokens [Xiao et al., 2024]: condensing the first token to the last layer's KV removes each layer's sink entry. Tables 4 and 6 ablate this choice together with the KV-construction modes; our final video model uses the exemption with *Reuse KV*.

B Ablation Tables

Tables 13–15 report the language-model analysis ablations of Section 4 as numbers, computed from the same evaluation data as the corresponding main-text results. The NVS source-layer ablation is Table 8 in Section 7.

Blending-coefficient sweep. The nearest-chunk blend is learned per head from a 0.5 initialization (Appendix A); a controlled sweep over fixed $\alpha \in \{0, 0.25, 0.5, 0.75, 1\}$ against the learned setting is left as an additional ablation.

Table 12: Full RULER results for the 760M models *without* gated attention (plain attention; accuracy, %; higher is better), in the same format as Table 11. Without the gate, LastKV variants overtake the full-cache Transformer in the 13-task mean at 8K–32K, and Shared KV is strongest at every length beyond 4K (32K mean: 30.0 vs. 20.6 for the full cache).

	Transformer (full cache)				LastKV Reuse (ours)				LastKV Per-layer				LastKV Shared			
Task	4K	8K	16K	32K	4K	8K	16K	32K	4K	8K	16K	32K	4K	8K	16K	32K
S-NIAH-1	99.6	97.2	94.0	55.8	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	99.4	97.8	88.8	**100.0**	**100.0**	99.0	97.8
S-NIAH-2	**100.0**	**100.0**	**100.0**	82.2	**100.0**	99.6	99.8	61.2	**100.0**	**100.0**	95.0	64.6	**100.0**	**100.0**	99.8	**91.6**
S-NIAH-3	69.2	30.0	7.2	4.6	70.2	75.2	49.2	17.2	69.6	65.8	30.6	21.4	**91.4**	**83.8**	**77.4**	**49.0**
MK-NIAH-1	**91.0**	**87.8**	**81.6**	49.6	74.8	68.8	52.0	45.4	77.4	81.2	64.8	45.6	85.0	73.0	69.4	**57.8**
MK-NIAH-2	21.6	26.2	12.6	1.6	**61.4**	**27.0**	**30.0**	2.8	32.8	19.0	8.8	**5.2**	10.2	14.6	10.6	1.4
MK-NIAH-3	4.4	1.4	0.4	0.0	**10.8**	**4.8**	**1.2**	**0.2**	1.8	0.0	0.2	0.0	5.8	1.8	**1.2**	**0.2**
MQ-NIAH	**75.1**	**60.0**	**37.8**	25.5	51.1	38.0	19.5	21.1	43.5	41.1	36.9	**29.5**	65.8	44.6	32.5	23.9
MV-NIAH	43.4	35.8	**31.1**	22.1	37.0	31.8	11.7	**26.0**	27.5	28.3	22.9	22.1	**58.1**	**38.0**	23.4	24.1
VT	9.3	3.1	**6.8**	0.8	1.6	0.1	0.1	0.1	6.7	0.4	2.8	0.6	**17.5**	**11.2**	4.0	**7.4**
CWE	**68.4**	**11.7**	0.2	0.1	15.5	8.3	0.6	**0.5**	1.6	0.2	0.0	0.1	0.4	0.2	**1.3**	0.1
FWE	5.3	2.5	0.2	0.3	1.0	9.7	16.3	0.7	20.7	13.3	3.1	2.2	**23.3**	**30.2**	**31.7**	**9.4**
QA-SQuAD	27.8	19.1	15.1	7.9	31.4	19.1	16.0	**8.4**	36.0	**22.2**	**19.1**	8.1	**39.5**	20.2	18.5	6.6
QA-HotpotQA	27.6	23.0	19.0	17.2	30.6	24.0	17.0	17.8	**34.2**	24.4	**22.2**	19.2	33.2	**30.4**	21.8	**21.0**
Mean (13 tasks)	**49.4**	38.3	31.2	20.6	45.0	39.0	31.8	23.2	42.5	38.1	31.1	23.6	48.5	**42.2**	**37.7**	**30.0**

Table 13: Chunk-size ablation (760M language model). Raw per-token validation loss, mean over positions $\geq$1000 and tail ($\geq$16,384). Smaller chunks write the last-layer cache more often and reach lower loss.

Chunk size C	1024	2048	4096	8192
Mean loss	–*	**2.5334**	2.5363	2.5448
Tail loss	–*	**2.5211**	2.5248	2.5337

*The raw per-token export for the 1024-chunk run is pending.

C Qualitative Results

Qualitative novel-view renderings and generated video frames will be included here; the diagnostic attention-map and boundary-loss visualization appears in Fig. 2.

D Reproducibility

All architectures and training recipes are fully specified in Tables 9 and 10, and the method is given as executable pseudocode in Algorithms 1, 2, and 3. Language models train on Long-Data-Collections with the Mistral tokenizer (32K vocabulary) and evaluate per-token loss on Books3 over 32K-token sequences and retrieval on RULER; NVS models train on Objaverse (object-level) and filtered DL3DV-10K (scene-level) with the benchmark splits held out; video models fine-tune Wan 2.1 T2V at 1.3B and 14B and evaluate on the official VBench protocol. Loss statistics in all tables are computed from raw (unsmoothed) per-token losses; figures apply time-weighted EMA (decay 0.99) for readability only. Results are single-run point estimates; seeds and multi-run variance are not yet reported.

E Limitations

Our experiments are controlled studies of last-layer KV-cache reuse, and several directions remain open. First, we do not report an end-to-end *serving* benchmark: decode latency, tokens/sec at inference, and maximum context under a fixed memory budget on an optimized serving stack. The research chunk-parallel implementation used for the from-scratch runs is also unoptimized — it re-reads the retained context per chunk and keeps per-layer full-length KV buffers — although the fused chunk-loop implementation used for the 8B conversion recovers most of the baseline's training throughput (Table 2).

Table 14: KV-construction ablation (760M language model). Raw per-token validation loss (mean over positions $\geq$1000; tail $\geq$16,384) for the three ways of building the retained cache, with the full-cache Transformer of the same family as reference. All three variants outperform the baseline; *Shared KV* is marginally best, and *Reuse KV* matches it within 7×10^{-4} while adding no extra projection.

	Reuse KV	Shared KV	Per-layer KV	Transformer
Mean loss	2.5373	**2.5366**	2.5375	2.5423
Tail loss	2.5250	**2.5246**	2.5250	2.5290

Table 15: Nearest-chunk refinement ablation (760M language model). Besides mean/tail loss, we quantify the chunk-boundary artifact of Fig. 2: the boundary spike is the mean loss over the 64 tokens following each 4096-token chunk boundary minus the mean loss elsewhere. The refinement removes the spike entirely ($+0.103 \rightarrow -0.003$ nats).

	Mean loss	Tail loss	Boundary spike (nats)
Without refinement	2.5450	2.5334	$+0.103$
With refinement	**2.5363**	**2.5248**	$\mathbf{-0.003}$

Second, we have not compared LastKV against state-of-the-art large language models, nor have we fine-tuned a state-of-the-art large language model into a LastKV variant. The language-modeling results therefore establish the behavior of the compact-cache architecture under matched training recipes rather than competitiveness with the strongest deployed LLM systems. For the same reason, we compare against a full-cache Transformer rather than head-to-head against other cross-layer sharing designs such as CLA, LCKV, or YOCO [Brandon et al., 2024, Wu and Tu, 2024, Sun et al., 2024]: those designs apply sharing at every position of a decoding sequence, so a controlled comparison requires aligning chunking, positional handling, and training procedures, which we leave to future work.

Third, our video experiments have not been scaled to larger video-generation models, longer training runs, or larger video datasets. This leaves open whether the same last-layer cache design continues to improve quality and efficiency at frontier video scale.

Fourth, we have not studied several potentially useful training and inference curricula. These include pretraining with a larger chunk size and then fine-tuning with a smaller chunk size, pretraining a dense Transformer and then continuing pretraining as a LastKV, combining last-layer cache reuse with token-selection policies, and using chunk-level cache writes during prefill while switching to token-level writes during autoregressive decoding. Each of these settings could improve the trade-off between parallelism, cache-write frequency, and inference memory, and we leave them to future work.

F Broader Impacts

The main broader impact of LastKV is inference efficiency. Transformer-style language models and autoregressive video generation models typically retain layer-specific KV caches for long histories. LastKV suggests an alternative in which older context can be represented by the last layer's KV cache and reused across future layers, so inference only needs to store one layer of KV per retained older chunk instead of a full per-layer KV cache. If the retained-cache reduction translates into system-level gains, it could make long-context LLM inference and long-horizon video generation cheaper, more energy efficient, and easier to serve on memory-constrained hardware.

Because the method is architecture-level and task-agnostic, the same cache-memory reduction could apply broadly to future LLMs and video generation models that use Transformer KV caches. The positive impact is lower serving cost and reduced hardware barriers for long-context or long-duration generation. At the same time, efficiency improvements can make both beneficial and harmful generative uses easier to deploy, so responsible deployment should follow the safeguards appropriate to the underlying LLM or video model, including access controls, misuse monitoring, and application-specific safety filters when generated content can affect people.