LOCATE, THEN CONTINUE: A FRAMEWORK FOR SHARED KV MEMORY IN HYBRID ATTENTION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Hybrid attention models combine linear attention for efficient long-context modeling with periodic full-attention calls for exact retrieval, yet it is unclear how far those calls, though occurring at different depths, can share computation and state, and what minimal degree of sharing exact retrieval still requires. We answer this question with a descriptive framework that views each full-attention call as writing to and reading from an associative memory. Within this framework, exact retrieval composes two read types: *associative lookup*, which locates a record by matching its identity, and *sequence continuation*, which copies the record verbatim by matching predecessor windows against the generated prefix. Reads of one type share a single key function and differ only in their queries, so two read types suffice for direct, chained, and multi-target retrieval regardless of read count, hop count, content length, or model depth. Conversely, under an empirically motivated role-stability premise, that queries alone cannot reliably switch one cache between lookup and continuation, one layer does not suffice, making two layers the minimum, typically ordered as lookup then continuation. Experiments on 0.5B and 1.3B hybrids support the framework's predictions: the shallow group specializes in lookup and the deep group in continuation, merging two continuation groups preserves retrieval, and a single shared layer fails to recover verbatim copying. The resulting design, Group Share, reduces independent full-attention layers from seven to two at 1.3B, saving 10.4% of parameters and 57.4% of total cache at 16K while maintaining general capabilities; it even yields higher retrieval accuracy at context lengths from 2K to 16K.

1 INTRODUCTION

Linear attention and state-space models reduce the cost of long sequences by compressing history into fixed-size recurrent states (Katharopoulos et al., 2020; Sun et al., 2023; Yang et al., 2024a;b; Gu & Dao, 2024; Dao & Gu, 2024), but this compression limits their ability to distinguish fine-grained associations and to reproduce content verbatim (Arora et al., 2024; Jelassi et al., 2024). Hybrid models therefore insert full attention (Vaswani et al., 2017) at selected depths, combining efficient state updates with direct access to per-token history (Lieber et al., 2024; Dong et al., 2024); Griffin and Samba pursue a similar combination with local attention (De et al., 2024; Ren et al., 2025). Periodic linear/full-attention patterns now appear in Qwen3-Next, Qwen3.5, Kimi K3, and MiniMax-01 (Qwen Team, 2025; 2026; Moonshot AI, 2026; MiniMax, 2025). In these designs, each full-attention call has its own layer, with independent parameters and a KV cache that grows with context length and batch size, so full-attention resources scale with the number of calls.

Cross-layer sharing can decouple these resources from depth. Universal Transformer and ALBERT share parameters across layers (Dehghani et al., 2019; Lan et al., 2020), as do the attention blocks of the hybrid Zamba models (Glorioso et al., 2024b;a); Cross-Layer Attention, MLKV, Hymba, and YOCO share K/V across layers (Brandon et al., 2024; Zuhri et al., 2025; Dong et al., 2024; Sun et al., 2024). MQA, GQA, and MLA (Shazeer, 2019; Ainslie et al., 2023; DeepSeek-AI, 2024) reduce KV storage within a layer, complementing cross-depth sharing. These methods show that sharing is feasible, but leave open how far it can extend without degrading exact retrieval. We describe exact retrieval as locating historical spans through query-specified associations and reproducing their content verbatim. We therefore ask: if full-attention calls at different depths may share a layer,

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

reading its KV cache with queries formed from their own states, what is the minimum number of independent full-attention layers that exact retrieval requires?

We answer this question with a descriptive framework for exact retrieval. Following the associative-memory view of attention (Ramsauer et al., 2021; Schlag et al., 2021), each full-attention call writes keys and values from its own state and reads them with a query formed from the reader's state; the reader may sit at another depth. The keys determine which positions can be matched, and the query selects among them. Exact retrieval then composes two read types: *associative lookup*, which locates a record by matching its identity, and *sequence continuation*, which copies the record by matching predecessor windows, akin to induction heads (Olsson et al., 2022). The framework reduces our question to a counting argument. Reads of one type use the same key function and differ only in their queries, so one layer per read type suffices regardless of read count. Conversely, we posit that queries alone cannot reliably switch readers of written K/V between the two types; under this premise, two layers are the minimum, typically ordered as lookup then continuation.

The framework prescribes **Group Share**: retain all depths and full-attention calls, but partition the calls into contiguous groups that share entire layers, including parameters and KV caches. The first call in each group writes the cache; later calls read it with their own queries (Figure 1).

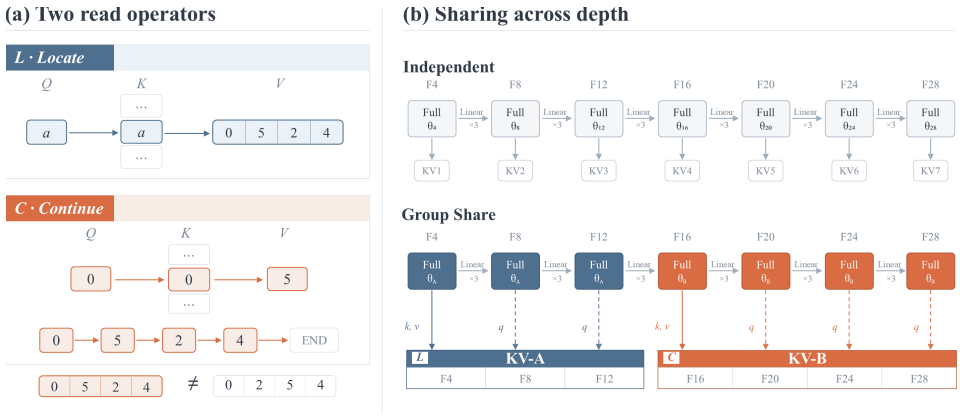

Figure 1: **Exact retrieval needs two read types, and Group Share gives each its own shared layer.** (a) A lookup read (*Locate*) matches the query against record identities and locates the record 0524. A continuation read (*Continue*) matches the current token against predecessor keys and reads the next token; repeated Continue reads recover the record in order, distinguishing 0524 from 0254. (b) Both architectures keep all seven full-attention calls ($\times 3$: three linear-attention layers between calls). The independent hybrid gives every call its own parameters and KV cache. Group Share partitions the calls into two contiguous groups, each sharing one parameter set (θ_A or θ_B) and one KV cache: the first call writes k, v (solid arrows), and later calls read the group cache with queries q from their current states (dashed arrows). Group A serves lookup (L) and group B serves continuation (C).

Our contributions are: **1. A descriptive framework for exact retrieval.** We describe full attention as writing to and reading from associative memory and exact retrieval as a composition of lookup and continuation. The number of independent full-attention layers follows the number of read types rather than reads: two suffice, and under role stability two are the minimum.

2. Tests of the framework's predictions. We test four predictions in 0.5B and 1.3B hybrids through group interventions, common-parent merging, single-cache controls, and K/V substitution.

3. Group Share. Sharing entire full-attention layers within two groups realizes the two-layer minimum; at 1.3B, it saves 10.4% of parameters and 57.4% of total cache at 16K while maintaining general capabilities, and even yields higher retrieval accuracy.

2 An Associative-Memory Framework for Exact Retrieval

This section states the framework; full statements and proofs appear in Appendix A.

2.1 WRITING AND READING IN HYBRID MODELS

A hybrid has L layers, predominantly linear attention with periodic full attention, e.g., Qwen3.5's `3 Linear + 1 Full`. We use Gated DeltaNet (GDN) (Yang et al., 2025) for linear layers, which compress history into fixed-size states. Because fixed-size recurrent states have limited capacity to distinguish among many associations and recover content verbatim, full attention operates at $s_1 < \cdots < s_M$, with call m receiving residual state h_{s_m}.

Each full-attention call writes and reads. **Writing** maps the state at depth s to a key–value pair at each position, forming an associative memory. **Reading** forms a query from the reader's own state at depth s' and retrieves from that memory:

$$k_j = \kappa(h_{s,j}), \quad v_j = \nu(h_{s,j}); \qquad q_t = \psi(h_{s',t}), \quad o_t = \sum_{j \le t} \mathrm{softmax}_j\big(q_t^\top k_j/\sqrt{d}\big) v_j.$$

Standard hybrids both write and read at each call, coupling M reads to M layers and caches. Queries must originate from the reader's own depth, but K/V need not. In the independently trained Qwen3.5-4B, several full-attention depths contribute to retrieval, and in the latter half, deeper states reading earlier K/V largely preserve the correct answer. How few independent layers can serve M reads?

A **read relation** $\mathcal{R}$ between reader state and written prefix specifies the target positions; we call the query–key matching that concentrates attention on them *addressing*. A sufficiently large score margin over non-target positions makes the softmax-weighted readout approximate the target value, with required margin growing logarithmically in context length. Exact reading thus depends on the key function's ability to separate matching positions.

2.2 TWO READ TYPES FOR EXACT RETRIEVAL

History consists of records (a_r, c_r), where a_r is a record identity and $c_r = (c_{r,1}, \ldots, c_{r,n_r})$ is its content, written token by token in the context. Exact retrieval locates the requested records, reproduces their contents verbatim, and terminates correctly. We consider **direct retrieval** D, which returns content given an identity (e.g., RULER single, UUID, and multi-key); **chained retrieval** Ch_h, which reaches a target through h within-record references (e.g., variable tracking); and **multi-target retrieval** MT_k, which returns k records (e.g., multi-value and multi-query).

Two read relations describe these tasks. **Associative lookup (L)** locates a target by matching identity; **sequence continuation (C)** recovers content verbatim by matching predecessor windows to retrieve the next content unit:

$$\mathcal{R}_L : \ \mathrm{id}(h_{s',t}) = a(j) \ \big[\wedge\, j = u_{a(j)}\big], \qquad \mathcal{R}_C : \ x_{j-w:j-1} = y_{k-w+1:k} \ \wedge \ \pi(j) = \pi(h_{s',t}).$$

Lookup keys encode record identity $a(j)$ and mark each record's entry u_r, the position immediately preceding $c_{r,1}$; a lookup query targets either all positions of the record or, with the bracketed condition, its entry alone. Continuation keys encode predecessor windows: the w tokens before j must match the generated prefix's suffix at the same retrieval progress π. The readout returns the next content unit or final record boundary.

Theorem 1 (Completeness, informal). Assume (A1, addressing) records are distinguishable by identity and some key function separates a record's positions, or its entry position alone, with a sufficient margin; (A2, retrieval progress) the generated suffix window and retrieval progress uniquely determine the next content position, e.g., the target has no repeated w-grams; and (A3, readout usability) a sufficiently accurate readout can be decoded into a content unit, record identity, or record entry and used to form the next query. Then all three retrieval classes are finite compositions of the operators Loc and Cont, each executing one lookup or continuation read:

$$D = \mathsf{Loc}\,;\, \mathsf{Cont}^{\,n_r+1}, \qquad Ch_h = \mathsf{Loc}^{\,h}\,;\, D, \qquad MT_k = D^{\,k},$$

where the semicolon denotes sequential execution.

Both read types are necessary. Under lookup alone, positions sharing a lookup key receive the same score for every query, so a record reaches the reader only through a few fixed averages of its values; at finite precision, such averages cannot distinguish all contents of sufficiently long records (with token-level values, `0 5 2 4` and `0 2 5 4` already coincide). Continuation alone cannot reach content separated from its identity by arbitrary-length text. Content *present in the KV cache* is thus not necessarily *addressable*.

2.3 From Read Counts to Group Counts

Proposition 1 (Composition changes only the query). In the programs of Theorem 1, all reads of the same type use the same key function; only their queries differ, as determined by the reader's current state. The entire program therefore needs only two key functions, independently of hop count, target count, content length, model depth, or context length.

Read count is supplied by the retained full-attention calls; independent layers need only cover read types, provided readers at other depths can form compatible queries and use the readouts.

Partition M calls into G contiguous groups sharing entire full-attention layers and caches (Section 3). Collapsing each maximal run of consecutive reads of the same type within a single token's forward pass yields a **type word** $w \in \{L, C\}^+$. In the programs of Theorem 1, direct retrieval reads its first content unit by locating the record's entry and continuing from it in the same pass (LC), and later units by continuation (C); a lookup hop gives L. Because identity precedes content, these tasks never need continuation before lookup within one pass; CL arises only when copied content must serve as an identity, e.g., retrieving "the attribute of the item after X in a list." Group types form a depth-ordered sequence T.

A token's forward pass traverses groups in depth order. With group types specified by T, execution depends on whether the program's type word is a subsequence of T. For example, $T = LC$ supports L, C, and LC but cannot execute CL because the C group follows the L group.

Theorem 2 (Two groups suffice). If w is a subsequence of T and each run fits within its assigned group's call budget, the program is realizable in the grouped architecture under the cross-depth sharing conditions above.

Premise H (Role stability). By Proposition 1, same-type reads share one cache and differ only in their queries. We posit the converse for jointly trained grouped models: within one group's cache, varying queries alone does not reliably switch readers between lookup and continuation, so each group serves one read type. H is a premise, not a consequence of the framework, since one cache can encode both relations in separate subspaces. Corollary 2 holds conditionally on H; Section 4.3 tests only H's prediction that single-cache models are uneven across the two read types.

Corollary 2 (Minimum group count under H). Under H, the subsequence condition is also necessary, so the minimum group count for a program family $\mathcal{P}$ is the length of the shortest common supersequence (SCS) of its type words: $G^*(\mathcal{P}) = \left|\mathrm{SCS}\{w(P) : P \in \mathcal{P}\}\right|$.

Direct, chained, and multi-target retrieval have type words L, C, and LC, whose SCS is LC. Two groups therefore suffice; under H, they are also necessary, and since LC must be a subsequence of T, they are ordered as lookup then continuation. If a single forward pass must also execute CL, the SCS becomes LCL or CLC, and a third group is needed. Greater depth adds reads within groups; with M full-attention calls, $G = 2$ saves $1 - 2/M$ of full-attention KV storage. Longer programs can write intermediate results as autoregressively generated tokens and execute over multiple steps using the same two groups.

Testable implications. (1) *Two read types* (Theorem 1, Corollary 2): trained groups typically develop a lookup-then-continuation functional profile across depth, with the shallower group specializing in lookup and the deeper group in continuation (Section 4.1). (2) *Cache count follows read types* (Proposition 1): same-type calls share one cache regardless of read count and write depth, so merging same-type groups should preserve retrieval (Section 4.2). (3) *Role stability* (H): a single cache retains the content, but single-cache models are uneven across the two read types (Section 4.3). (4) *Cross-depth sharing conditions*: later readers use the content of reused K/V (Section 4.4).

3 Group Share

Group Share partitions M full-attention calls into G contiguous groups, each sharing an entire full-attention layer and one KV cache. All call positions and the linear layers' independent parameters and states are retained: M controls read count, and G controls independent resources. We use $G = 2$: by Theorem 2, lookup followed by continuation covers direct, chained, and multi-target retrieval without increasing group count with depth. $G = 1$ is Full Share; $G = M$ is the standard independent-call hybrid. We write Gk for $G = k$ and Fn for the full-attention call at layer n.

From the write–read perspective of Section 2.1, Group Share changes where K/V are written. Each group's first call generates K/V from its residual state and appends them to the group cache. Later calls form queries from their own residual states and read this cache without generating K/V again. All calls in a group share normalization, Q/K/V/O projections, attention gating, and the FFN; different inputs still yield different queries. During training, gradients from all reads jointly update the shared parameters and K/V generation paths, allowing writing and cross-depth reading to co-adapt.

Group Share reduces full-attention parameters and KV storage to G/M of the standard allocation, and K/V projections and writes per token from M to G. All M calls still execute query projection, attention reading, output projection, and FFN, leaving global-read computation unchanged.

4 Testing the Framework's Predictions

We test the four implications of Section 2.3 in order at two scales. We use the 0.5B ablations of Section 4.2 and the 1.3B Baseline and G2 of Section 5. Retrieval uses six RULER NIAH tasks (Hsieh et al., 2024) with greedy decoding, scored by whole-target exact match for single/multi-key and complete-target-value recall for multi-value/query.

4.1 Two Groups Implement the Two Read Types

In frozen 0.5B/1.3B G2 models, we zero selected attention-branch outputs, retaining residual paths and FFNs. On 13 RULER tasks at 2K (50 examples each), functional impairment is the increase in mean next-correct-token loss across answer positions, conditioned on correct prefixes. We also measure loss changes on 128 FineWeb validation sequences, separately for copyable tokens, whose bigram with the preceding token occurs earlier in the sequence, and other tokens (Table 1).

Table 1: **Functional interventions on the two groups.** Entries are loss increases (nats) after disabling a group; larger values indicate stronger impairment. Bold marks the larger impairment within each model and diagnostic column. Column groups indicate diagnostic emphasis; retrieval tasks use both read types. A = F4/F8/F12; B = F16/F20/F24, plus F28 at 1.3B.

		Repeated lookup		Continuation		LM control
Model	Disabled	Multi-value	Multi-query	UUID	Copyable tokens	Other tokens
0.5B G2	Group A	**1.57**	**1.60**	0.48	0.14	0.06
0.5B G2	Group B	0.79	0.79	**3.45**	**1.26**	0.15
1.3B G2	Group A	**1.12**	**1.03**	0.03	0.04	0.03
1.3B G2	Group B	0.75	0.77	**3.26**	**0.91**	0.13

At 1.3B, disabling A raises Multi-value/query loss by 1.12/1.03 nats, compared with 0.75/0.77 for B. Disabling B raises UUID loss by 3.26 nats, compared with 0.03 for A, and affects copyable FineWeb tokens more than other tokens (0.91 versus 0.13 nats). The same pattern at 0.5B supports a division of labor between shallow lookup and deeper continuation.

Continuation's key function κ_C must encode predecessor windows. We linearly predict current/preceding-token input embeddings from pre-RoPE keys at write sites and measure held-out nearest-neighbor top-1 accuracy. Across three 192/64 train/test splits of 256 sequences, the mean preceding-to-current-token accuracy ratio is 0.29 for A and 0.68 for B at 0.5B, versus 0.40 and 0.83 at 1.3B. Greater predecessor decodability in B agrees with κ_C, although probes do not identify the addressing subspace.

4.2 Cache Count Follows Read Types

The 0.5B architectures in Table 2 share the $[3\,\mathrm{GDN} + 1\,\mathrm{Full}] \times 6$ backbone, six call sites, linear layers, and residual paths. Only independent full-attention parameter sets P, key sources S_K, and value sources S_V vary. All train for approximately 2.0B tokens at 2K with identical data order, tokenizer, initialization rules, and optimization. G2/G3 use $\mathrm{AAA\,BBB/AA\,BB\,CC}$. Two one-sided variants retain G1's single parameter set but refresh K or V at F16 (Table 2).

Table 2: **Sharing ablations under matched training budgets.** Retrieval: unweighted mean over six 2K RULER tasks, 300 examples each. Full KV: theoretical single-sequence BF16 storage. Zamba-like shares only full-attention parameters on our backbone; shallow-write G2 writes B's K/V at F4 instead of F16.

Architecture	$P/S_K/S_V$	Params (M)	Full KV (MiB)	FineWeb loss	Retrieval (%)	UUID (%)	Multi-key (%)
Baseline	6/6/6	501.66	12	3.3168	56.97	79.67	30.00
Zamba-like	1/6/6	451.03	12	3.3370	56.85	78.33	26.67
KV-only	6/1/1	499.70	2	3.3179	52.39	39.00	30.33
G1	1/1/1	451.03	2	3.3253	55.38	62.33	33.67
G2	2/2/2	461.15	4	3.3258	57.76	79.00	37.33
G3	3/3/3	471.28	6	3.3270	60.58	87.67	28.67
Shallow-write G2	2/2/2	461.15	4	3.3146	59.72	81.00	27.00
K-refresh/V-share	1/2/1	451.03	3	3.3218	56.69	64.67	30.67
K-share/V-refresh	1/1/2	451.03	3	3.3377	34.86	11.67	24.33

Parameter-only Zamba-like nearly matches Baseline (78.33% versus 79.67% UUID): six reads need not require six parameter sets. G2 reduces six caches to two and reaches 79.00% UUID and 37.33% Multi-key. Proposition 1 further predicts that cache count depends on read types rather than write depth. *Shallow-write G2* keeps G2's parameter sets, groups, and calls, but B's projections write its K/V at F4 from A's residual input; B still reads at F16 and later. Its UUID accuracy is 81.00%: continuation K/V can be written shallowly and read deeply.

Neither one-sided refresh matches G2: K-only refresh gives 64.67% UUID, close to G1, while V-only refresh gives 11.67%. A cache in Section 2 is a pair (κ, ν) written by one parameter set and shared by its readers. Replacing one side creates no new group and may break the pairing of keys and values, motivating an entire full-attention layer per group.

G3 scores higher on UUID (87.67%) but lower on Multi-key (28.67%). Differences among models trained from scratch conflate architecture with optimization, and all FineWeb losses lie within 3.3146–3.3377, so retrieval gains do not track LM loss. We therefore test Proposition 1 directly by merging two groups of the same type within one pretrained model.

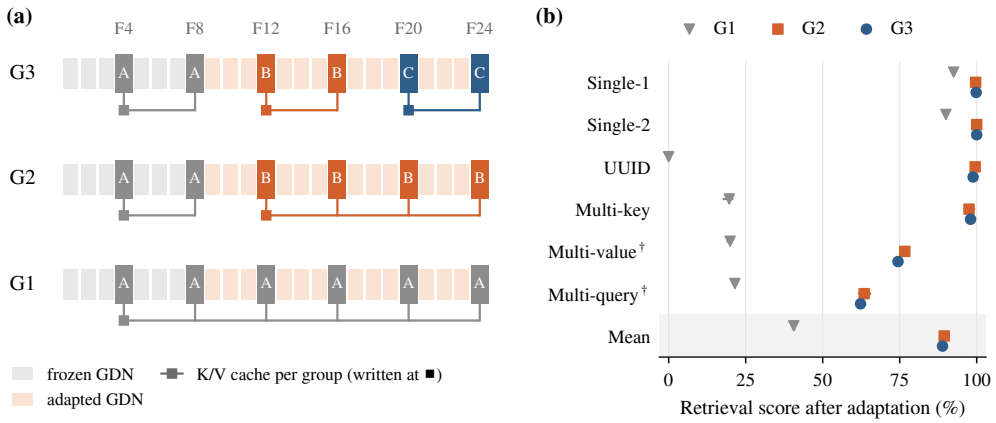

Figure 2: **Paired G3-to-G2/G1 replacement from a common parent.** (a) All retain depth and six full-attention calls. Letters denote complete parameter sets. Each group writes one cache at its first call and reads it throughout. Full-attention parameters are frozen; only the twelve GDN layers after layer 9 are adapted. (b) Free generation on 1,200 held-out questions. Markers show means across three adaptation repeats; horizontal bars show their ranges. † denotes complete-target-value recall; other metrics are exact match.

From one pretrained G3, we construct G2 by merging AA BB CC into AA BBBB: F20/F24 use B's entire full-attention layer and read K/V written at F12 (Figure 2a). C and B both perform continuation, so the merge introduces no new read type. Depth, all calls, and existing GDN layers are preserved.

Full-attention parameters, embeddings, and final normalization are frozen; merged and original models adapt only the twelve GDN layers after layer 9, using identical data, order, and update budgets.

After adaptation, G2 and G3 reach six-task means of 89.47% and 88.90%, a paired difference of +0.56 percentage points (95% interval $[+0.12, +1.01]$), with nearly overlapping task results (Figure 2b). Direct merging causes errors on 30 UUID and multi-key questions that the original model answered correctly; all are recovered in every adaptation run. B's K/V thus supports the reads previously served by C.

4.3 A Single Cache Supports the Two Read Types Unevenly

KV-only and G1 keep a single cache; their UUID accuracy falls to 39.00% and 62.33% (Table 2). The deficit lies in addressing rather than content. On 400 UUID questions, gold addressing, which specifies the source token to copy, raises G1 from 64.2% to 96.8% and KV-only from 38.0% to 98.8%. A region oracle, which restricts attention to the target UUID span and renormalizes it, splits the remaining gap into two *addressing components*: locating the target span and aligning within it. The two models fail on different components (Table 3).

Merging into one group from the common parent does not recover continuation. This G1 uses A's parameters and F4 K/V for all six calls (Figure 2a). Direct merging reduces all six tasks to zero. With the twelve later GDN layers adapted, G1's K/V remains that of group A and only its queries change; single-key tasks recover above 90%, but UUID remains at 0.00% in every repeat and the six-task mean is 40.60%. Adapting all eighteen GDN layers, which also changes the K/V, raises G1's mean to 51.24%, still with 0.00% UUID; G2 and G3 reach 90.22% and 90.78%.

Chained retrieval shows the same contrast. In a pointer-chasing task, random-length distractors make every hop a lookup, and intermediate words are externalized before a final UUID, so the chain executes over generation steps. After one fine-tuning run per model, G2 stays at 97–98% for $h = 1$–4, whereas G1 falls from 96.5% to 89.0% (gap at $h = 4$: 8.0 points, 95% interval $[3.0, 13.0]$), failing on lookups rather than UUID copying: 56 of G1's 59 errors involve a wrong intermediate word.

Table 3: **Single-cache diagnostics at 0.5B.** (a) Free-generation accuracy after adaptation. (b) From-scratch UUID losses: Span = region oracle minus native; Within = gold addressing minus region oracle. Protocols are in Appendix E.

(a) Accuracy (%, ↑)

Setting	G1	G2
Merged: UUID	0.0	97.2
Pointer: 4-hop UUID	89.0	97.0

(b) Addressing loss (pp, ↓)

Model	Span	Within
G1	29.0	3.5
KV-only	11.5	49.3
G2	10.8	7.3

Table 3 separates adapted task accuracy from addressing losses. Merged G1 retains 99–100% single-key lookup accuracy but fails on UUID continuation. On the pointer task, teacher-forced UUID-token accuracy is at least 99.98% for both G1 and G2, while G1 makes more lookup errors. From-scratch G1's gap is primarily in span selection; KV-only's is in within-span alignment. These asymmetric deficits are consistent with H in the tested 0.5B settings.

4.4 Later Readers Use the Content of Reused K/V

Cross-depth sharing also requires readers to form compatible queries from their own states and use the retrieved content. Section 4.2 adapts GDN layers before F12, allowing B's write state to change. To test reuse of existing K/V, we fix both A/B generation paths and adapt only the nine GDN layers after F12. For identical inputs, G2's A/B K/V are byte-identical before and after adaptation. G2 reaches 74.86% mean retrieval versus 72.97% for matched-budget G3, with a paired 95% interval of $[+1.28, +2.52]$ percentage points. All 21 UUID questions damaged by merging recover in every repeat. Recovery thus comes from adapting later readers to fixed K/V.

We test actual content use at the same adapted checkpoints. Holding the original input, answer-start state, and GDN recurrent states fixed, we replace only B's K/V at the target value and following period with those for a new value. Across 288 questions covering six tasks, complete-new-target generation

reaches 75.12%, close to 76.85% when naturally editing the input; V-only substitution reaches 19.10%. New values enter through local K/V and are used by the recovered readers. Under V-only substitution, K still encodes the original value's predecessor windows, misaligning continuation's successor matching with the new content.

Table 4: **Resources and native 2K quality after 15B-token pretraining.** One pretraining run per model; 300 examples per retrieval task. General tasks report zero-shot accuracy; multi-value and multi-query report complete-target-value recall; other retrieval tasks report exact match.

Metric	Baseline	Group	Metric	Baseline	Group
Parameters	1.308B	1.172B	**General tasks (accuracy, %)**		
Full KV caches	7	2	HellaSwag	35.94	35.36
FineWeb loss	2.9581	2.9787	PIQA	70.02	70.24
FineWeb perplexity	19.26	19.66	WinoGrande	53.43	52.88
RULER NIAH (%)			ARC-Easy	48.86	49.54
single-1	100.00	100.00	ARC-Challenge	21.25	20.90
single-2	100.00	100.00	LAMBADA	41.67	41.80
single-3 (UUID)	96.33	94.33	BoolQ	58.17	58.04
multi-key-1	47.00	56.67			
multi-value	23.33	37.42			
multi-query	28.00	36.25			
Six-task mean	65.78	70.78			

5 Group Share at Scale

We compare Baseline and G2 at 1.3B under matched native 2K pretraining and matched 16K continued pretraining, assessing quality, exact retrieval, and resource use.

5.1 Experimental Setup

Both use a 28-layer Qwen3.5 backbone, $[3\,\text{Gated DeltaNet} + 1\,\text{Full}] \times 7$. Baseline has seven independent full-attention layers and caches; Group Share groups layers $\{4, 8, 12\}$ and $\{16, 20, 24, 28\}$. Depth, width, and call positions are identical. Stage one trains from random initialization for 15B tokens at 2K, matching FineWeb data (Penedo et al., 2024), order, tokenizer, seed, and optimization. Stage two uses the same 0.5B-token continuation data (80% contiguous 16K segments, 20% 2K replay), retaining native RoPE. Appendix B gives implementation details, hyperparameters, and measurement protocols.

Native 2K evaluation covers FineWeb, seven zero-shot tasks (Zellers et al., 2019; Bisk et al., 2020; Sakaguchi et al., 2020; Clark et al., 2018; Paperno et al., 2016; Clark et al., 2019) (Table 4) with lm-evaluation-harness texts and scoring (Gao et al., 2024), and the six RULER tasks of Section 4 on 300 fixed examples per task and length. After continuation, we evaluate loss in 2K position bins over the first 16K tokens of 128 held-out long FineWeb documents, plus retrieval at eight lengths from 2K to 16K. We also measure post-prefill total cache (full-attention K/V plus Gated DeltaNet states) and peak GPU memory during 64-token generation.

5.2 Quality and Resources under Matched Pretraining Budgets

Group Share reduces parameters from 1.308B to 1.172B (10.40%) and full-attention caches from seven to two (71.43% storage savings). FineWeb perplexity rises by 2.09% relatively; all seven general-task accuracy differences remain within one percentage point. Mean retrieval improves from 65.78% to 70.78%, mainly through multi-key/value/query (Table 4).

5.3 Long-Context Quality and Memory Savings

After matched 16K continuation, token-loss gaps are 0.019–0.026 nats across eight position bins, without steadily widening toward the sequence end (Figure 3a). Group Share achieves a higher mean

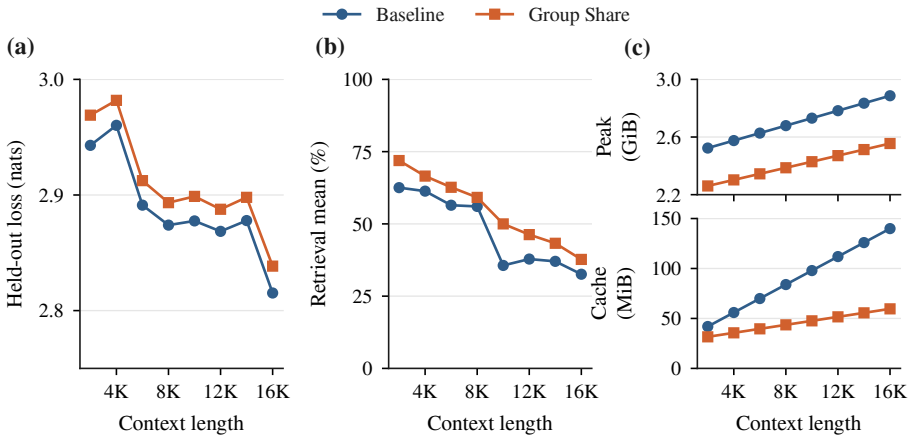

Figure 3: **Quality and memory after matched 16K continued pretraining.** (a) Token loss in 2K position bins over 128 long FineWeb documents; the horizontal axis gives bin endpoints. (b) Unweighted six-task retrieval mean, with 1,800 fixed examples per length and model (300 per task). (c) Peak GPU memory during 64-token generation (top) and total cache after prefill (bottom), including full-attention KV caches and Gated DeltaNet recurrent states.

retrieval score at all eight context lengths, with gains of 2.85–12.64 percentage points (Figure 3b). Both decline with length, to 37.99% (Group Share) and 32.94% (Baseline) at 16K.

Because full-attention KV storage grows linearly with length while Gated DeltaNet state size remains constant, total-cache savings increase from 24.6% at 2K to 57.4% at 16K (Figure 3c). Peak GPU memory is lower at every tested length, by 10.4% at 2K and 11.5% at 16K.

6 Conclusion and Limitations

We described exact retrieval in hybrid attention models as associative memory accessed by two read types, lookup and continuation. Because reads of one type share a key function and differ only in their queries, the number of independent full-attention layers follows the number of read types rather than the number of reads: two layers suffice, and under role stability they are the minimum, typically ordered as lookup then continuation. Interventions, common-parent merging, single-cache controls, and K/V substitution in 0.5B and 1.3B hybrids support this view. Group Share realizes this minimum: at 1.3B with $M = 7$, it saves 71.43% $(1-2/M)$ of full-attention parameters and KV storage and 57.4% of total cache at 16K while maintaining general capabilities, and even yields higher retrieval accuracy.

Several limitations remain. The sufficiency of two groups is an expressivity result; the minimum group count depends on the empirical premise H, because one cache can support both read types in terms of expressivity. Tests of H's prediction are concentrated on synthetic retrieval at 0.5B, with only group-level interventions at 1.3B. Experiments cover 0.5B and 1.3B scales, with 15B pretraining tokens at 1.3B and one pretraining run per architecture. Group Share also has 10.4% fewer parameters than Baseline at 1.3B, so its 2.09% relative perplexity difference (Section 5.2) cannot be separated from the effect of parameter count. Single-cache variants store half of G2's full-attention KV, so the one- versus two-group comparisons do not control storage. Group Share retains all full-attention calls and does not reduce attention computation, and group boundaries have not been systematically searched. Long-context evaluation mainly uses synthetic RULER tasks up to 16K, and whether the framework extends to open-domain tasks such as long-document reasoning remains open. The framework also makes a prediction not tested here: retrieval that requires continuation before lookup within one forward pass, with type word CL, needs a third group, which would provide a further test of H.

Reproducibility Statement

We provide the complete experimental code as anonymized supplementary material and will publicly release the code and all pretrained checkpoints (0.5B ablation architectures, 1.3B Baseline and Group Share before and after 16K continuation, and the adapted models of Sections 4.3–4.4) upon publication. The code includes: (i) the Group Share implementation on the Qwen3.5-style hybrid backbone and every sharing variant in Tables 2 and 5 (Appendix B.1); (ii) pretraining and continued-pretraining configurations with data preparation and ordering scripts, under which all architectures at a scale consume identical batches verified by per-step hashes (Appendices B.2 and B.5); (iii) RULER example generation with fixed seeds, scoring scripts, and the lm-evaluation-harness configuration (Section 5.1; Appendix B.3); (iv) intervention, probing, region-oracle, and gold-addressing scripts (Appendices C–E); (v) data generators, splits, and adaptation scripts for the common-parent replacement, content-substitution, and multi-hop experiments (Appendices F–G); (vi) the Qwen3.5-4B-Base intervention scripts, which use only the public base checkpoint (Appendix I); and (vii) system-measurement scripts (Appendix B.4). A README maps every table and figure to its script and configuration. Hyperparameters appear in Appendices B.2, F.1, and F.3; software versions and hardware in Appendix B.4; and complete statements and proofs of all theoretical results in Appendix A.

AI Use Statement

We used ChatGPT and Claude Code in this work, as follows.

- **Research design.** We developed the main research ideas and experimental designs. AI tools proposed one experiment, adaptation with fixed K/V generation (Section 4.4), which we adopted after review.
- **Code and data.** AI tools wrote much of the experimental and visualization code and generated synthetic test data, such as the inputs for the probing experiments. We reviewed all code and data.
- **Writing.** AI tools drafted the related-work paragraph of Section 1 and parts of the experimental descriptions in Section 4, and assisted with translation and language editing. We checked every drafted passage.
- **Literature.** AI tools helped us search for and read related work.

We verified every experimental result and drew all conclusions. We take full responsibility for the content of this paper.

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 *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://aclanthology.org/2023.emnlp-main.298/.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=LY3ukUANko.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. URL https://arxiv.org/abs/1911.11641.

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. doi: 10.52202/079017-2758. URL https://arxiv.org/abs/2405.12981.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *Proceedings*

of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics, 2019. URL https://aclanthology.org/N19-1300/.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? Try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018. URL https://arxiv.org/abs/1803.05457.

Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2405.21060.

Soham De, Samuel L. Smith, Anushan Fernando, Aleksandar Botev, George Cristian-Muraru, Albert Gu, Ruba Haroun, Leonard Berrada, Yutian Chen, Srivatsan Srinivasan, Guillaume Desjardins, Arnaud Doucet, David Budden, Yee Whye Teh, Razvan Pascanu, Nando De Freitas, and Caglar Gulcehre. Griffin: Mixing gated linear recurrences with local attention for efficient language models. *arXiv preprint arXiv:2402.19427*, 2024. URL https://arxiv.org/abs/2402.19427.

DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024. URL https://arxiv.org/abs/2405.04434.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1807.03819.

Xin Dong, Yonggan Fu, Shizhe Diao, Wonmin Byeon, Zijia Chen, Ameya Sunil Mahabaleshwarkar, Shih-Yang Liu, Matthijs Van Keirsbilck, Min-Hung Chen, Yoshi Suhara, Yingyan Lin, Jan Kautz, and Pavlo Molchanov. Hymba: A hybrid-head architecture for small language models. *arXiv preprint arXiv:2411.13676*, 2024. URL https://arxiv.org/abs/2411.13676.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 2024. URL https://github.com/EleutherAI/lm-evaluation-harness.

Paolo Glorioso, Quentin Anthony, Yury Tokpanov, Anna Golubeva, Vasudev Shyam, James Whittington, Jonathan Pilault, and Beren Millidge. The Zamba2 suite: Technical report. *arXiv preprint arXiv:2411.15242*, 2024a. URL https://arxiv.org/abs/2411.15242.

Paolo Glorioso, Quentin Anthony, Yury Tokpanov, James Whittington, Jonathan Pilault, Adam Ibrahim, and Beren Millidge. Zamba: A compact 7b ssm hybrid model. *arXiv preprint arXiv:2405.16712*, 2024b. URL https://arxiv.org/abs/2405.16712.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2312.00752.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kloBbc76Sy.

Samy Jelassi, David Brandfonbrener, Sham M. Kakade, and Eran Malach. Repeat after me: Transformers are better than state space models at copying. In *International Conference on Machine Learning*, pp. 21502–21521, 2024. URL https://proceedings.mlr.press/v235/jelassi24a.html.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, 2020. URL https://arxiv.org/abs/2006.16236.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1909.11942.

Opher Lieber, Barak Lenz, Hofit Bata, Gal Cohen, Jhonathan Osin, Itay Dalmedigos, Erez Safahi, Shaked Meirom, Yonatan Belinkov, Shai Shalev-Shwartz, Omri Abend, Raz Alon, Tomer Asida, Amir Bergman, Roman Glozman, Michael Gokhman, Avashalom Manevich, Nir Ratner, Noam Rozen, Erez Shwartz, Mor Zusman, and Yoav Shoham. Jamba: A hybrid transformer-mamba language model. *arXiv preprint arXiv:2403.19887*, 2024. URL https://arxiv.org/abs/2403.19887.

MiniMax. MiniMax-01: Scaling foundation models with lightning attention. *arXiv preprint arXiv:2501.08313*, 2025. URL https://arxiv.org/abs/2501.08313.

Moonshot AI. Kimi K3 technical report. Technical report, Moonshot AI, 2026. URL https://github.com/MoonshotAI/Kimi-K3/blob/3cb39dfd32e51c3328e2e4b4af21341247d06c43/k3_tech_report.pdf. Accessed 2026-08-09.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads. *Transformer Circuits Thread*, 2022. URL https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016. URL https://aclanthology.org/P16-1144/.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2406.17557.

Qwen Team. Qwen3-Next: Towards ultimate training & inference efficiency. Qwen Blog, 2025. URL https://qwen.ai/blog?id=4074cca80393150c248e508aa62983f9cb7d27cd. Accessed 2026-09-18.

Qwen Team. Qwen3.5: Towards native multimodal agents. Qwen Blog, February 2026. URL https://qwen.ai/blog?id=qwen3.5. Accessed 2026-08-09.

Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, Victor Greiff, David Kreil, Michael Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. Hopfield networks is all you need. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2008.02217.

Liliang Ren, Yang Liu, Yadong Lu, Yelong Shen, Chen Liang, and Weizhu Chen. Samba: Simple hybrid state space models for efficient unlimited context language modeling. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=bIlnpVM4bc.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020. URL https://arxiv.org/abs/1907.10641.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, 2021. URL https://arxiv.org/abs/2102.11174.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019. URL https://arxiv.org/abs/1911.02150.

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. URL https://arxiv.org/abs/2307.08621.

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. *arXiv preprint arXiv:2405.05254*, 2024. URL https://arxiv.org/abs/2405.05254.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://arxiv.org/abs/1706.03762.

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*, 2024a. URL https://arxiv.org/abs/2312.06635.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Advances in Neural Information Processing Systems*, volume 37, 2024b. URL https://arxiv.org/abs/2406.06484.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2412.06464.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019. URL https://aclanthology.org/P19-1472/.

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*, pp. 5531–5540, 2025. doi: 10.18653/v1/2025.findings-naacl.305. URL https://aclanthology.org/2025.findings-naacl.305/.

APPENDIX OVERVIEW

Appendices follow the order of the main text. Each main-text claim is self-contained; the appendix gives proofs, protocols, and supporting diagnostics.

Appendix	Content	Supports
A	Full statements and proofs; cross-depth sharing conditions	Section 2
B	Implementation, training, evaluation, and system-measurement protocols; training curves	Sections 3 and 5
C	Per-call interventions, reads within one group, key probes	Section 4.1
D	Sharing ablations: value sources, write depth, one-sided refresh, addressing gap	Sections 4.2–4.3
E	Settings of Table 3 (Premise H)	Section 4.3
F	Common-parent replacement and multi-hop chained retrieval	Sections 4.2–4.3
G	Fixed K/V generation and content substitution	Section 4.4
H	Per-task long-context curves	Section 5.3
I	Multi-depth reading and cross-depth reuse in Qwen3.5-4B	Section 2.1

A FULL STATEMENTS AND PROOFS FOR SECTION 2

We use Section 2's notation. Scores are $s_j = q_t^\top k_j/\sqrt{d}$, and the reader state is $h_{s',t}$.

A.1 ACCURACY OF SOFT READS

A read retrieves positions $J^*(t) = \{j : \mathcal{R}(h_{s',t}, x_{\le j})\}$; exact reading requires attention to concentrate on $J^*(t)$.

Lemma A.1 (Accuracy of soft reads). Suppose a single target position j^* satisfies $s_{j^*} \ge s_j + \Delta$ for all $j \ne j^*$, there are n candidate positions, and $\|v_j\| \le V$. Then $\|o_t - v_{j^*}\| \le 2V(n-1)e^{-\Delta}$.

Proof.

$$1 - p_{j^*} = \frac{\sum_{j \ne j^*} e^{s_j}}{e^{s_{j^*}} + \sum_{j \ne j^*} e^{s_j}} \le \frac{(n-1)e^{-\Delta}}{1 + (n-1)e^{-\Delta}} \le (n-1)e^{-\Delta}.$$

Since $o_t - v_{j^*} = \sum_{j \ne j^*} p_j (v_j - v_{j^*})$ and $\|v_j - v_{j^*}\| \le 2V$, we obtain $\|o_t - v_{j^*}\| \le 2V(1-p_{j^*}) \le 2V(n-1)e^{-\Delta}$. □

Thus $\Delta \ge \log(n-1) + \log(2V/\varepsilon)$ guarantees readout error at most ε: the required margin grows logarithmically with context length.

A.2 CONDITIONS AND PROOF OF THEOREM 1

Let $a(j)$ identify the record containing position j, and let u_r be record r's entry position, immediately preceding $c_{r,1}$. Lookup keys encode $a(j)$ together with the entry mark $\mathbb{1}[j = u_{a(j)}]$, e.g., $\kappa_L(h_j) = [\phi(a(j)); \mathbb{1}[j = u_{a(j)}]]$: a query with zero weight on the mark targets all positions of the record, and a positive weight λ raises the entry above them by λ. Each hop of chained retrieval is assumed to need only the next record's identity.

- **A1 (Addressing).** The record identity of every content position and of the entry position is determined by the prefix available at writing; different records are distinguishable; and a key function exists that gives $\mathcal{R}_L$ the margin in Lemma A.1, both over a record's positions and over its entry position alone.
- **A2 (Retrieval progress).** For target content, the window ending at the entry or at the last generated unit, together with progress π, uniquely determines the next position, e.g., the target contains no repeated w-grams, or progress disambiguates repeats.

- **A3 (Readout usability).** A readout within tolerance can be decoded into a content unit, record identity, or record entry; retained computation can then update progress and form the next query.

The final step of Cont^{n_r+1} retrieves the record boundary. We induct on generation step i, showing that each step outputs correctly and retains the identity and progress needed for the next.

Direct retrieval.

- *Step 1 (type word LC).* The request state encodes a_r. By A1, a lookup query separates the entry position u_r by margin Δ, so by Lemma A.1 the readout is within ε of v_{u_r}. By A3, the reader decodes the entry, sets progress $\pi = (r, 0)$, and, in the same forward pass, forms the continuation query for the position after u_r. A2 identifies this position as that of $c_{r,1}$, and Lemma A.1 and A3 recover $c_{r,1}$. The state then records progress $\pi = (r, 1)$.
- *Step* $i + 1$. The state contains $y_{1:i} = c_{r,1:i}$ and progress (r, i). A2 uniquely identifies the position satisfying $\mathcal{R}_C$, namely that of $c_{r,i+1}$. Lemma A.1 and A3 recover $c_{r,i+1}$ and update progress to $(r, i + 1)$.
- *Termination.* Step $n_r + 1$ retrieves the record boundary and terminates.

Chained retrieval. Hop i forms its query from the identity retrieved at hop $i - 1$, using $\mathcal{R}_L$ without the entry condition. A1 and A3 ensure that each hop retrieves the next identity; after h hops, the problem reduces to direct retrieval.

Multi-target retrieval. Progress additionally tracks completed targets, and direct retrieval repeats k times.

Every step applies one of the two operators, so all three classes are finite compositions of them. □

A.3 Both Read Types Are Necessary

Lemma A.2 (Indistinguishability). Let reader state be z_i, query $q_i = Q_i(z_i)$, readout $r_i = \mathrm{Read}(q_i; \mathcal{M})$, and update $z_{i+1} = U_i(z_i, r_i)$. If $z_0 = z'_0$ and $\mathrm{Read}(q; \mathcal{M}) = \mathrm{Read}(q; \mathcal{M}')$ for every admissible q, then states and outputs remain identical after any finite number of steps.

Proof. By induction, $z_i = z'_i$ implies $q_i = q'_i$, hence $r_i = r'_i$ and $z_{i+1} = z'_{i+1}$. If two histories require different correct outputs, both cannot be satisfied; randomized outputs likewise have identical conditional distributions. □

The lemma distinguishes content being present in the KV cache from being addressable: if no admissible query distinguishes it from other content, the reader cannot use it.

Proposition A.1 (Minimality). Neither associative lookup alone nor sequence continuation alone can solve all three retrieval classes.

Lookup alone. Lookup keys depend only on identity and the entry mark, so positions sharing a key form one class and receive the same score for every query. Fix the number of records and the reader's initial state, and let values take 2^β levels per coordinate. Every readout is then $\sum_r w_r(q)\bar{v}_r$, where $\bar{v}_r$ averages the values in class r (at most two classes per record, plus one for unlabeled positions) and $w_r(q)$ depends only on the query, the keys, and the class sizes. In a context of length T, each $\bar{v}_r$ takes at most $(T2^\beta + 1)^d$ values, polynomially many in T, whereas a target record of length n has exponentially many possible contents. For sufficiently large n, two histories with different target contents and equal class sizes therefore have identical averages and identical readouts for every query, and by Lemma A.2 one of them receives a wrong token. With token-level values, `0 5 2 4` and `0 2 5 4` already coincide: both within-record averages are $(v_0 + v_5 + v_2 + v_4)/4$. Distinguishing them requires matching the predecessor relation, i.e., continuation.

Continuation alone. Insert distractor text of random length between identity and content. For any fixed w, the w tokens before the content start belong to the distractor and are unrelated to the requested identity. Thus $\mathcal{R}_C$ cannot select the content start from the request, whereas A1 allows $\mathcal{R}_L$ to select the record's entry by identity. □

A.4 Proof of Proposition 1

Theorem 1's construction uses only $\mathcal{R}_L$ and $\mathcal{R}_C$. Each constrains only the writer's key function: κ_L encodes identity and entries, and κ_C encodes predecessor windows and progress cues. Reads differ only in their queries, formed from current states. One κ_L satisfying A1 therefore supports all lookup reads, and one κ_C satisfying A2 supports all continuation reads. The number of key functions is independent of read count, and hence of hops, targets, and content length. It is also independent of depth and context length: Lemma A.1's margin condition depends only on the key function and the logarithm of context length. □

A.5 Cross-Depth Sharing Conditions

Proposition 1 fixes the key functions; sharing a cache across depth additionally requires that a reader at another depth can *use* what it reads. We state the conditions used by Theorem 2.

Read–process interchange. Fix the current state h. The shared cache returns positional content v_j, and suppose the content a reader needs is $f(h, v_j)$ for a single function f shared across positions and contexts. Let $p_j \geq 0$, $\sum_j p_j = 1$, and $\bar{v} = \sum_j p_j v_j$. If $f(h, \cdot)$ is twice differentiable on the convex hull of the contents with second derivative bounded by H, the integral Taylor remainder and $\sum_j p_j(v_j - \bar{v}) = 0$ give

$$\Big\|\sum_j p_j f(h, v_j) - f(h, \bar{v})\Big\| \leq \frac{H}{2} \sum_j p_j \|v_j - \bar{v}\|^2.$$

Interchange is exact when p selects one content unit (without requiring linear f), and for arbitrary p when f is affine in its content argument. Thus "process each position, store a new cache, then read" can become "read existing content, then process it with the current state."

From ideal interchange to actual replacement. Let an additional interface have content $v_{C,j} = f(h, v_j) + e_j$ and addressing p^C, the existing interface use p^B, and the network implement $\widehat{f}$ with $\|\widehat{f}(h, \bar{v}_B) - f(h, \bar{v}_B)\| \leq \epsilon_{\text{impl}}$, where $\bar{v}_B = \sum_j p_j^B v_j$ and $V_{\max} = \max_j \|v_{C,j}\|$. The triangle inequality yields

$$\Big\|\sum_j p_j^C v_{C,j} - \widehat{f}(h, \bar{v}_B)\Big\| \leq V_{\max}\|p^C - p^B\|_1 + \sum_j p_j^B \|e_j\| + \frac{H}{2} \sum_j p_j^B \|v_j - \bar{v}_B\|^2 + \epsilon_{\text{impl}},$$

i.e., addressing difference, representation error, interchange remainder, and implementation error. h and v_j must be available to the retained computation, and f cannot be chosen per test context.

Propagation across depth. Let E_ℓ be the error in the relevant state after segment ℓ, ϵ_ℓ the local replacement error, and L_ℓ a Lipschitz bound of the reference state map. Then $E_{\ell+1} \leq L_\ell E_\ell + \epsilon_\ell$, so $E_m \leq \sum_{i<m} \epsilon_i \prod_{k=i+1}^{m-1} L_k$; e.g., $L_\ell \leq \rho < 1$ and $\epsilon_\ell \leq \epsilon$ give $E_m \leq \epsilon/(1-\rho)$. Small local errors alone therefore do not ensure bounded error at arbitrary depth; we do not claim that trained models are contractions.

Sharing conditions (used in Theorem 2). (S1) readers at every depth in a group can form queries compatible with the group's key function; (S2) the retrieved content is usable by the reader's retained computation within the bounds above; (S3) the resulting state errors propagate boundedly. These conditions are about realizability; Sections 4.3–4.4 test them through actual adaptation (Appendix F) and content substitution (Appendix G).

A.6 Proofs of Theorem 2 and Corollary 2

Proof of Theorem 2. Choose a subsequence embedding $g_1 < \cdots < g_\ell$ of $w = \tau'_1 \cdots \tau'_\ell$ in T. Assign run i to readers at different depths within group g_i. Proposition 1 permits a common key function; conditions S1–S3 permit compatible queries and usable readouts. The residual stream transfers state between runs. A3 and the induction in Theorem 1 establish correct output. This direction does not require H.

Proof of Corollary 2. Under H, each group serves only one read type, so groups execute reads according to their roles. A token's forward pass advances monotonically through depth and can

use groups only in order. The program's type word must therefore be a subsequence of T. To accommodate all words in $\mathcal{P}$, T must be a common supersequence, giving $G \geq |\mathrm{SCS}|$. Conversely, choosing a shortest common supersequence as the group order, with sufficient calls for each run, realizes every program, giving $G^* = |\mathrm{SCS}|$. □

Corollary A.1 (Alternating type words). Under H and with sufficient read budgets, if a family contains both alternating words of length n, one starting with L and one with C, then $G^* = n + 1$.

- *Lower bound.* A length-n sequence has only one length-n subsequence, itself. The two words differ, so $|\mathrm{SCS}| \geq n + 1$.
- *Upper bound.* Prefixing C to the word starting with L gives an alternating sequence of length $n + 1$ that contains both words and every alternating word of length at most n.

For $n = 2$, supporting both `LC` and `CL` within one forward pass requires three groups. □

A.7 Two Groups Suffice with Autoregressive Execution

Condition E (State transfer across generation steps). At each split, the current state can encode the record identity, retrieval progress, and intermediate results needed later into finitely many intermediate tokens; when fed back, retained computation can reconstruct the corresponding query conditions. The output format permits these tokens and specifies the final answer's location, and A1–A3 hold for contexts extended by generated tokens.

Proposition A.2 (Autoregressive execution). Under E, any finite L/C program can execute on an `LC` grouped architecture in finitely many generation steps. Split at C→L transitions, when a group's read budget is exhausted, or when a token must be emitted; splitting only at C→L gives $1 + \#\{i : \tau_i = C, \tau_{i+1} = L\}$ segments.

Proof. Every nonempty segment has form $L^a C^b$, with each type's read count within its group's capacity, so by Theorem 2 each segment executes on the same `LC` architecture. Induct over segments: E restores the information needed at each segment's start, and readers form new queries over the same two K/V groups. The program has finitely many reads and each segment completes at least one, so execution terminates. New tokens merely append entries to the two caches. □

A.8 Scope of the Results

- Theorem 1, Proposition 1, Theorem 2, and Propositions A.1–A.2 concern expressivity: they describe realizable computations, not a guarantee that training learns them.
- Corollary 2 depends on the empirical premise H, which cannot be derived from expressivity. Take $k_j = [\kappa_L(h_j); \kappa_C(h_j)]$: a reader using $q = [q_L; 0]$ or $[0; q_C]$ obtains only the $\mathcal{R}_L$ or $\mathcal{R}_C$ term, with Lemma A.1's margin unchanged. One cache can therefore support both read types in principle; layer 12 of the 1.3B Baseline performs both in different heads (Appendix C). H concerns jointly trained grouped models; its prediction for single-cache models is tested in Sections 4.2–4.3 (summary in Appendix E). The prediction that executing CL within one forward pass needs a third group is not tested here.
- This construction does not save storage: each key type occupies its own subspace, so one cache must be as wide as the two combined. Even if a widened single cache supported both read types, this would affect only whether the group is the right counting unit, not the $1 - 2/M$ storage savings.
- Intermediate autoregressive tokens still consume time and cache storage.
- These results concern exact retrieval as defined in Section 2, not the effect of sharing on language-modeling loss.

B Implementation, Training, and Evaluation Protocols

B.1 Implementation Details

The 1.3B models have hidden size 1280 and FFN size 5632. Full attention uses ten query heads and two K/V heads, each with dimension 128. Sections 2–3 omit multi-head structure: the query-head count n_q is an integer multiple of n_{kv}, and query head a reads K/V head $\gamma(a) = \lfloor (a-1)/(n_q/n_{kv}) \rfloor +$

1; concatenated head outputs pass through the group's shared gate and output projection. Group g applies RoPE to keys generated at its first call and caches them; each call applies RoPE to queries generated at its own depth, with positional encoding identical to Baseline. Attention normalization, gating, output projection, FFN normalization, and the FFN all belong to the group's shared parameter set and are reused by every call in the group.

B.2 Training Hyperparameters

1.3B. Stage one starts from random initialization at length 2K, with a global batch of 65,536 tokens, for 228,881 steps and 14,999,945,216 tokens. We use AdamW ($\beta = (0.9, 0.95)$, weight decay 0.1), peak learning rate 2.5×10^{-4}, 1% warmup, cosine decay to 10% of the peak, and BF16. Stage two starts from both final checkpoints and uses the same 0.5B-token FineWeb continued-pretraining data (80% contiguous 16K segments from individual documents, 20% 2K replay), with peak learning rate 2.5×10^{-5} for 7,630 steps. Data, order, tokenizer, initialization, random seeds, and optimization are identical between models in both stages.

0.5B ablations. All architectures in Table 2 and the shifted-boundary variant of Appendix D.2 train for 32,552 steps, approximately 2B tokens, with seed 42. Per-step audits confirm that every architecture's training-batch hashes match Baseline's within each scale.

B.3 Retrieval Evaluation Protocol

The six generative RULER tasks are Single NIAH-1/2/3, Multi-key NIAH-1, Multi-value NIAH, and Multi-query NIAH. Single NIAH-1 uses repeated digit strings and Single NIAH-3 uses UUIDs. Multi-value and Multi-query use complete-target-value recall; numeric Multi-key scores the first complete digit string; others use whole-target exact match. The original prompts for these tasks do not end in a colon; the evaluator appends one ASCII `:`. The prompts for variable tracking, CWE, FWE, and the two QA tasks already end in `:`, so this suffix would produce `::`; we therefore use these five tasks only in teacher-forced interventions (Appendix C), which supply the correct answer prefix. `niah_multikey_2/3` are at the capability floor (0–6%) at both scales and are excluded from generative results. Prompt lengths at 2K range from 1,327 to 1,842 Qwen tokens.

Frozen-model diagnostics. Appendices C–D use each architecture's final pretraining checkpoint without adaptation. They use forward passes without caching: a space token is fixed after the prompt colon, scoring starts at the answer's first character, and generation stops at the first error, so scores are not directly comparable with Table 2. Each intervention changes only the specified K/V sources, readouts, or attention scope. Region oracles and gold addressing use task labels and characterize attainable upper bounds, not native capability.

B.4 System Measurement

System measurements in Section 5.3 use 16K continued-pretraining checkpoints and four same-GPU paired runs on an A100-40GB with native PyTorch SDPA and batch 1. Incremental caching and full recomputation produce identical argmax predictions at 64 positions and identical eight-token greedy continuations (maximum absolute BF16 hidden-state difference 0.125). We use PyTorch 2.13.0+cu130, Transformers 5.14.1, and BF16; Gated DeltaNet uses the Transformers PyTorch implementation without `causal_conv1d` or fused kernels. Results characterize relative changes on this path, not deployment stacks such as vLLM. Native SDPA prefill exceeds 40GB at batch 4 beyond 8K, so Figure 3 reports batch 1.

FlashAttention-2 check. On native 2K checkpoints with FlashAttention-2 on four A100-SXM4-40GB GPUs (four order-balanced pairs, prompt lengths 128–2,048, batch 1/4), total-cache reduction matches the main text (24.60% at 2K), and peak allocated memory decreases by 10.0%/8.8% at 2K for batch 1/4. Cached-decoding throughput is $1.057\times$ Baseline (range [1.033,1.083]) at batch 1 and $1.066\times$ ([1.058,1.070]) at batch 4; prefill latency is essentially unchanged ($1.006\times$, $1.000\times$).

B.5 Training and Validation Curves

Training loss uses a trailing 200-step mean; validation loss is unsmoothed and uses fixed sets of 64 (0.5B) and 512 (1.3B) FineWeb sequences of length 2K, not interchangeable with the final evaluations.

Top rows use a piecewise-linear, nonuniform loss scale; bottom rows show the final third of training on linear gap axes. The 1.3B curves exclude the 16K continuation.

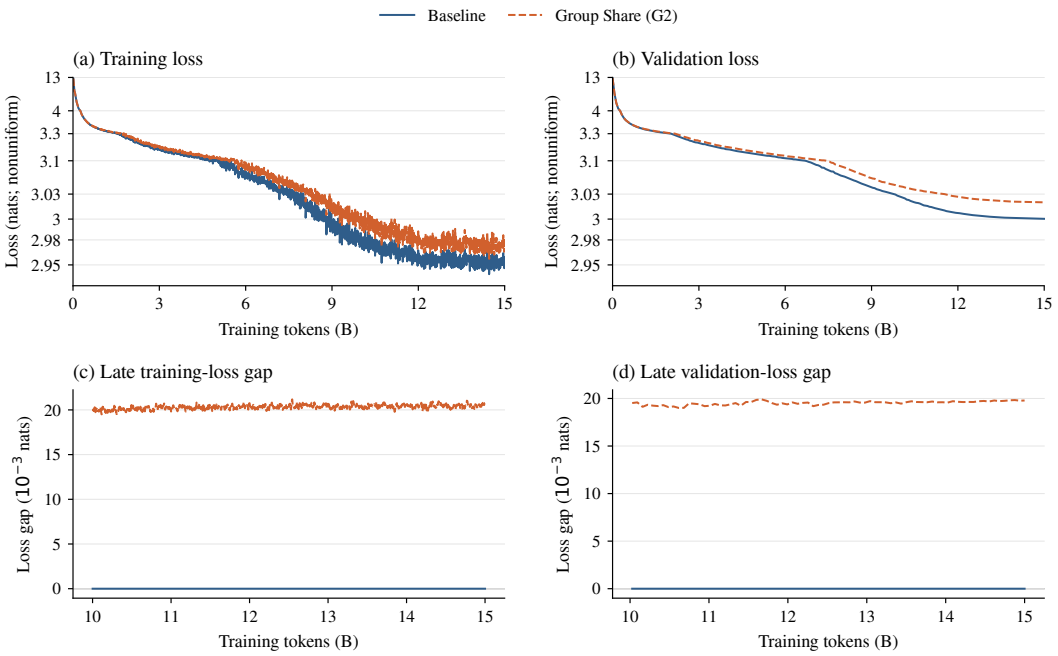

Figure 4: **Training histories for the 1.3B comparison** (228,881 steps, about 15B tokens). (a–b) Training and validation losses on nonuniform vertical axes; (c–d) late-training gaps relative to Baseline on linear axes.

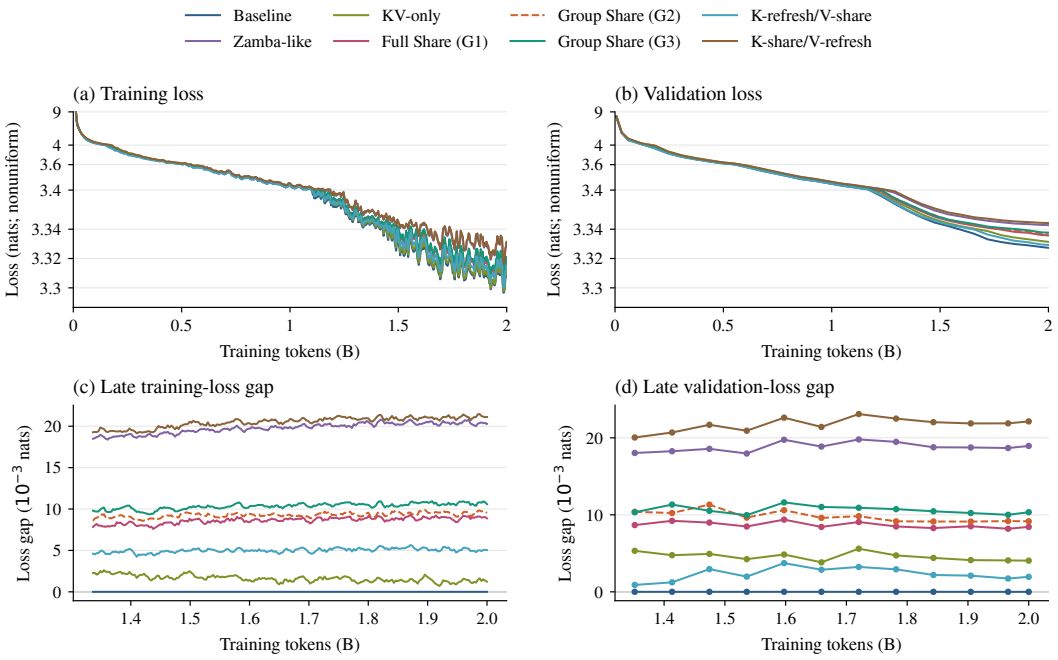

Figure 5: **Training histories for all eight 0.5B ablation architectures** (32,552 steps, about 2B tokens). Panels follow Figure 4.

Over the final third of training, Group Share's validation-loss gap relative to Baseline is 0.0190–0.0199 nats at 1.3B (77 measurements) and 0.0091–0.0113 nats at 0.5B (12 measurements); Zamba-like's 0.5B gap is 0.0180–0.0198 nats. Gaps are stable late in the observed runs, but we do not establish their behavior under longer budgets. G2's retrieval gains in Table 2 therefore do not coincide with lower language-modeling loss.

C Interventions and Probes for Section 4.1

C.1 Intervention Protocol

We define three interventions on each full-attention branch, retaining residual paths, FFNs, and other calls: zero its output everywhere (remove the read); zero it only at prompt-context positions ("context"); or retain only one call within a group, zeroing its other calls ("only"). Calls in a shared group use the same cache; zeroing a call does not change the cache. Retrieval uses the first 50 examples of each of 13 RULER tasks at 2K: NIAH single-1/2/3, multi-key-1/2/3, multi-value, multi-query, variable tracking (VT), CWE, FWE, and SQuAD/HotpotQA QA. We supply the correct answer prefix, compute next-correct-token loss averaged over answer positions, and report increases over the native model, so larger values indicate stronger impairment; this is not free generation. LM evaluation uses 128 FineWeb validation sequences of length 2K; a token is copyable when its bigram with the preceding token occurs earlier in the same sequence. Tables omit single-2, CWE, and the two QA tasks, whose patterns match those shown; complete results for all 13 tasks and for all six models are included in the supplementary material.

C.2 Per-Call Functional Maps

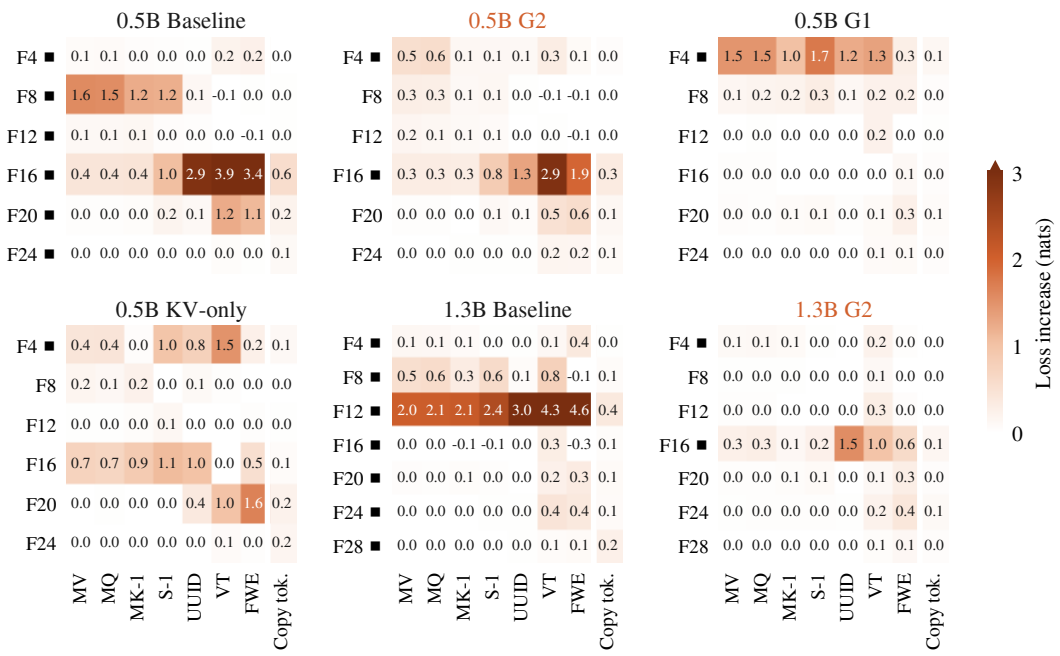

Figure 6: Per-call interventions in six models. Each cell gives the increase in mean correct-answer-token loss (nats) after removing one full-attention call; the final column gives the increase on copyable FineWeb tokens. Filled squares mark calls that write K/V.

Three patterns recur across the six models. First, reads at context positions have limited retrieval effects (largest 1.56, A in 1.3B Baseline on MK-3; most below 0.4), unlike Qwen3.5-4B layers 12/16 (Appendix I). Second, decisive reads occur at write sites. Baseline selects at F8 and copies at F16 at 0.5B, and does both at F12 at 1.3B (in different heads). G2 copies at F16; G1 concentrates both functions at F4. KV-only does not write at F16: its F16 read accesses F4 cache, removing it raises

UUID loss by only 1.00, and native UUID exact match is 36%. Third, cross-depth reads contribute to language modeling: in G2, retaining only B's final read approaches the full-group removal effect. Read-count dependence also differs between groups: in 0.5B G2, retaining only B's F16 raises UUID loss by just 0.14 nats, while retaining only A's F4 still raises multi-value/query loss by 0.79/0.77 (Table D.1).

Table D.1: Per-site interventions in 0.5B G2. A = F4/F8/F12; B = F16/F20/F24. Native FineWeb loss: 3.3814; copyable tokens: 1.3008; other tokens: 3.8551.

Condition	S-1	UUID	MK-1	MK-2	MK-3	MV	MQ	VT	FWE	Copy	Other
A context	0.33	0.08	0.04	0.15	1.34	0.06	0.07	0.19	−0.17	–	–
B context	0.17	0.04	0.13	0.03	0.08	0.02	0.03	0.16	0.02	–	–
Remove F4	0.13	0.11	0.13	0.36	1.32	0.53	0.57	0.30	0.09	+0.04	+0.01
Remove F8	0.13	0.01	0.11	0.32	0.47	0.28	0.26	−0.06	−0.06	+0.01	+0.00
Remove F12	0.08	0.01	0.10	0.25	0.20	0.15	0.13	0.02	−0.08	+0.01	+0.00
Remove F16	0.84	1.31	0.30	0.32	1.94	0.30	0.30	2.92	1.94	+0.34	+0.02
Remove F20	0.11	0.06	−0.01	−0.06	0.10	0.02	0.03	0.50	0.57	+0.13	+0.01
Remove F24	0.02	0.01	0.01	−0.01	−0.07	0.01	0.01	0.17	0.16	+0.08	+0.01
Remove A	1.15	0.48	1.13	0.52	1.96	1.57	1.60	0.61	−0.08	+0.14	+0.06
A: only F4	0.33	0.03	0.42	0.45	0.90	0.79	0.77	−0.07	−0.20	+0.03	+0.01
A: only F8	0.31	0.17	0.39	0.47	1.61	1.09	1.13	0.43	−0.11	+0.05	+0.02
A: only F12	0.51	0.34	0.65	0.48	1.76	1.41	1.48	0.50	0.12	+0.09	+0.04
Remove B	1.24	3.45	0.80	0.55	2.81	0.79	0.79	4.79	6.28	+1.26	+0.15
B: only F16	0.17	0.14	0.03	−0.06	0.12	0.06	0.07	1.13	1.32	+0.38	+0.06
B: only F20	0.85	1.41	0.33	0.34	1.99	0.32	0.33	3.21	2.64	+0.52	+0.04
B: only F24	1.13	2.91	0.60	0.41	2.52	0.52	0.52	4.37	4.71	+0.81	+0.08
Remove all	2.59	3.67	2.64	1.01	2.74	2.28	2.32	4.67	6.07	+1.31	+0.23

Table D.2: Per-site interventions in 1.3B G2. A = F4/F8/F12; B = F16/F20/F24/F28. Native FineWeb loss: 3.0331; copyable tokens: 1.1519; other tokens: 3.4614.

Condition	S-1	UUID	MK-1	MK-2	MK-3	MV	MQ	VT	FWE	Copy	Other
A context	0.11	0.01	0.04	0.38	0.55	0.06	0.05	0.69	0.13	–	–
B context	0.04	0.02	0.01	0.21	0.12	0.04	0.03	−0.17	0.18	–	–
Remove F4	0.02	0.01	0.09	0.63	0.31	0.10	0.11	0.22	0.03	+0.01	+0.01
Remove F8	0.00	0.00	0.02	0.28	0.01	0.02	0.02	0.12	−0.01	+0.01	+0.00
Remove F12	0.00	0.00	−0.03	0.42	0.08	0.03	0.02	0.29	0.04	+0.01	+0.00
Remove F16	0.24	1.54	0.13	0.62	2.12	0.25	0.27	0.98	0.61	+0.15	+0.01
Remove F20	0.08	0.01	0.05	0.02	0.03	0.03	0.03	0.12	0.28	+0.04	+0.00
Remove F24	0.03	0.02	0.03	−0.03	0.03	0.02	0.02	0.22	0.40	+0.10	+0.01
Remove F28	0.00	0.00	0.00	−0.01	−0.01	0.00	0.00	0.06	0.12	+0.04	+0.01
Remove A	0.67	0.03	0.63	1.10	0.65	1.12	1.03	1.15	0.11	+0.04	+0.03
A: only F4	0.02	0.00	0.04	0.92	0.16	0.14	0.13	0.59	0.06	+0.02	+0.01
A: only F8	0.07	0.01	0.24	1.03	0.50	0.49	0.47	0.68	0.08	+0.02	+0.02
A: only F12	0.12	0.01	0.32	0.86	0.33	0.38	0.36	0.49	0.03	+0.02	+0.02
Remove B	1.09	3.26	0.85	0.90	2.94	0.75	0.77	4.98	7.36	+0.91	+0.13
B: only F16	0.27	0.51	0.15	0.11	0.87	0.16	0.17	1.69	2.76	+0.51	+0.09
B: only F20	0.38	1.81	0.20	0.60	2.20	0.33	0.36	1.75	1.92	+0.46	+0.06
B: only F24	0.65	2.14	0.49	0.82	2.42	0.46	0.48	2.63	2.37	+0.35	+0.04
B: only F28	1.06	3.19	0.84	0.92	2.86	0.75	0.76	4.79	6.63	+0.71	+0.08
Remove all	2.70	3.40	2.60	1.46	2.88	2.42	2.44	4.84	7.30	+0.96	+0.17

C.3 All Readers in a Group Use the Shared Cache

On Single NIAH questions with target values in the first 20% of the prompt, all three B calls allocate substantial attention to the target, F20 highest (21 repeated-digit and 20 UUID questions in panel a). Read restoration in panel b uses 300 questions per task, 600 in total. Subtracting target-token readouts from all three calls at every answer step reduces exact copying to zero; restoring them sequentially at F16/F20/F24 yields $21/300, 171/300, 224/300$ (repeated digits) and $117/300, 206/300, 231/300$ (UUIDs). Every read contributes, consistent with Proposition 1: reads from one cache differ only in their queries. In a separate teacher-forced set (100 questions per task; 162 natively correct for G2),

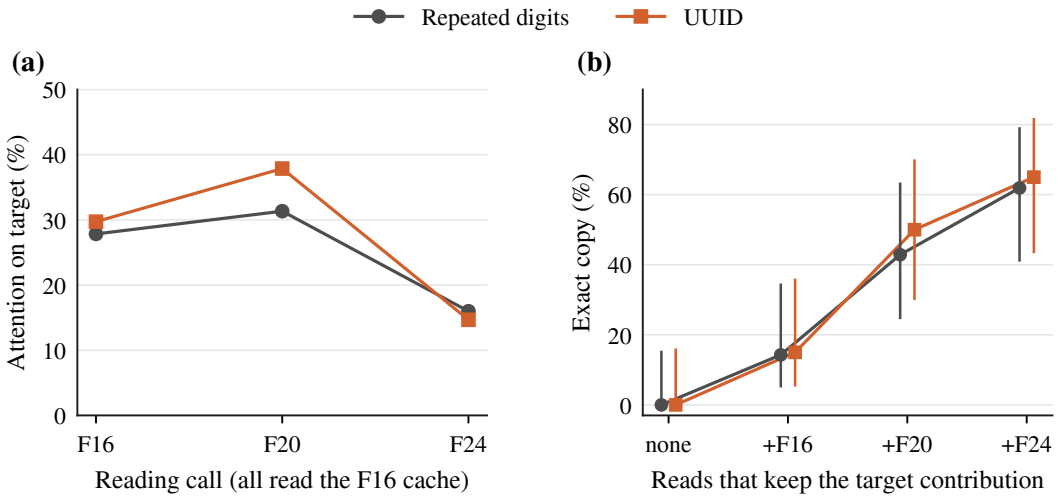

Figure 7: All three calls in 0.5B G2's B group read the cache written at F16. (a) Attention allocated to the target value at each call (21 repeated-digit and 20 UUID questions). (b) Exact copying as target-value readouts are restored sequentially at F16, F20, and F24 (300 questions per task); bars show 95% Wilson intervals.

removing target readouts at F16/F20 breaks 160 of these sequences; restoring only F24's native query rescues none, but letting F24 read values aligned with answer tokens rescues 30/160, and shifting these values by one token reduces recovery to 0. The cache retains token-level content that later readers use when read positions are aligned.

Readout compatibility. Using values generated from other depths' states with the receiving call's native attention weights, gating, and output projection, G2 shows high within-group and low cross-group cosine similarity to native contributions (within-minus-cross differences 0.801/0.822/0.775 on repeated digits, natural digits, and UUIDs), versus near zero for Baseline, G1, and K-refresh/V-share (Figure 8). The block structure reflects how learned readouts are organized by the architecture's groups.

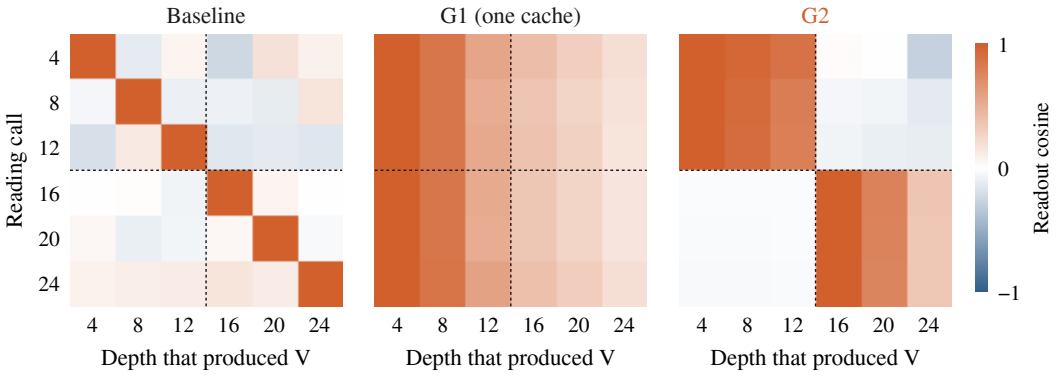

Figure 8: Readout compatibility on UUIDs ($n = 100$). Each cell gives the cosine similarity between a call's contribution using values from another depth and its native contribution. Dashed lines mark the F12/F16 group boundary.

C.4 Linear Probes of Keys

Ridge regression ($\lambda = 10$) predicts the input embedding of token $t - k$, $k = 0, 1, 2$, from each write site's pre-RoPE key (concatenated GQA heads). For all eight models, we sample every fourth

position in 256 FineWeb sequences of length 2K, each the first half of a distinct 4K validation block. Seeds 42/43/44 each split these blocks into 192 training and 64 test sequences. The nearest-neighbor candidate vocabulary is the fixed union of target token IDs over all selected sequences and offsets. We report the mean of the three split-specific accuracy ratios; the splits share the same source blocks. Residual-layer probes retain the original 24 sequences, 18/6 train/test split, and test vocabulary. In residual states of all six models, preceding-token decodability saturates at 0.26–0.31 by layers 2–3 and $t-2$ never exceeds 0.14, so preceding-token information is available from layer 3 onward; whether keys encode it depends on the write site's addressing role rather than depth (Table D.3 and Appendix D.2). Projecting out key subspaces predictable from preceding context collapses all tasks, while projecting out current-token subspaces has almost no effect; linear subspaces do not separate the two addressing mechanisms, which limits what probes can establish.

Table D.3: Ratio of preceding-token to current-token decodability from keys at write sites. Means over three 192/64 train/test splits of 256 sequences. Blank cells: calls that do not write K/V. Bold: copying sites.

Model	F4	F8	F12	F16	F20	F24	F28
0.5B Baseline	0.26	0.34	0.38	**0.72**	0.59	0.48	–
0.5B G2	0.29			**0.68**			–
0.5B G1	0.43						–
0.5B KV-only	0.41						–
1.3B Baseline	0.38	0.39	**0.90**	0.73	0.70	0.56	0.41
1.3B G2	0.40			**0.83**			

D Sharing Ablations for Sections 4.2–4.3

D.1 Two Value Sources Can Replace Six within One Checkpoint

To exclude cross-model training differences, we freeze Zamba-like and change only its six value sources from `V4/V8/V12/V16/V20/V24` to `V4/V4/V4/V16/V16/V16`, retaining parameters, key sources, and read count.

Task	Native six sources	Two sources	Difference (95% CI)
Single NIAH-1	100	100	0
Single NIAH-2	100	100	0
UUID	75	84	$+9.0$ $[+2.0, +16.0]$
Multi-key NIAH-1	26	27	$+1.0$ $[-6.0, +9.0]$
Multi-value NIAH	19.0	21.5	$+2.5$ $[+0.5, +4.5]$
Multi-query NIAH	18.8	21.2	$+2.5$ $[+1.0, +4.3]$
Macro-average	**56.5**	**59.0**	$\mathbf{+2.50}$ $\mathbf{[+0.67, +4.33]}$

No task's point estimate decreases. Conversely, rerouting a frozen Baseline (F4/F8/F12 to K/V written at F4 and F16/F20/F24 to F16) without adaptation drops six-task RULER from 56.50% to 16.71% and UUID from 80.5% to 9.0% (paired difference -71.5 points, 95% CI $[-76.0, -67.0]$): independently trained readers are co-adapted to their own K/V, which is why Section 4.2 adapts GDN layers after merging.

D.2 Write-Depth Variants

We train two 0.5B write-depth variants with the same recipe, retaining two complete parameter sets and two caches. *Shallow-write G2* (Section 4.2) retains G2's groups but writes B's K/V at F4, using B's own projections on the same residual input as A. *Shifted-boundary G2* instead moves the group boundary: A=$\{\mathrm{F4}\}$ and B=$\{\mathrm{F8}, \ldots, \mathrm{F24}\}$, writing B at F8. Both have 461.15M parameters and two caches, as G2. Key probes (Appendix C.4) at the variants' write sites give, for shifted-boundary G2, ratios of 0.25 at F4 (lookup) and 0.53 at F8 (continuation), consistent with native G2's 0.29/0.68: copying specialization follows B's write site rather than mid-depth. In shallow-write G2 A/B ratios are 0.44/0.42, yet UUID remains 81.00%; the write-depth conclusion rests on retrieval results, not on the coarse probe.

Table 5: **Write-depth variants.** Six 2K RULER tasks, 300 examples each, with Table 2's scoring. Shifted-boundary G2 also reduces A's read count; shallow-write G2 retains G2's groups.

Model	B K/V write	FineWeb loss	Retrieval mean (%)	UUID (%)	Multi-key (%)
G2	F16	3.3258	57.76	79.00	37.33
Shallow-write G2	F4	3.3146	59.72	81.00	27.00
Shifted-boundary G2	F8	3.3213	53.96	88.33	23.67
G1	No second cache	3.3253	55.38	62.33	33.67
KV-only	No second cache	3.3179	52.39	39.00	30.33

D.3 One-Sided Refresh: An Addressing Gap Rather than Missing Content

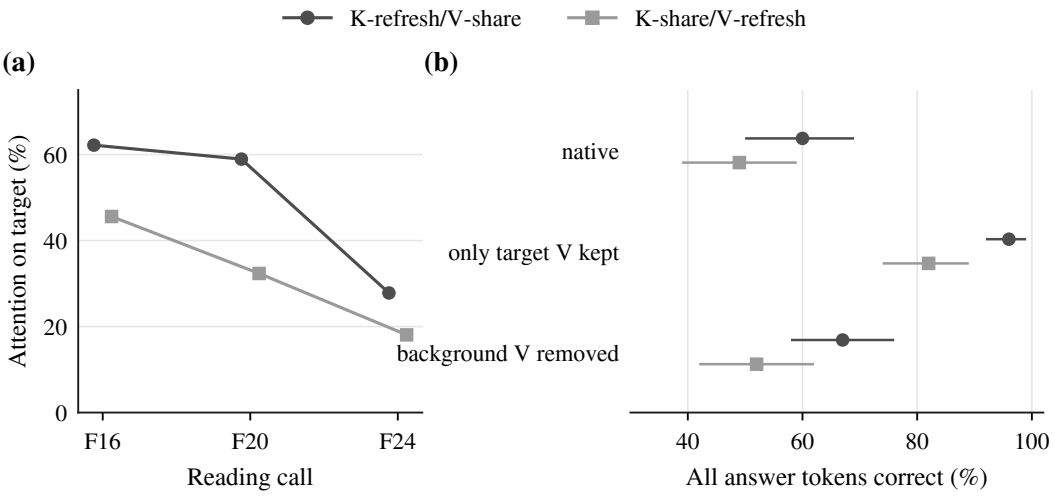

Figure 9: UUID diagnostics for the two one-sided refresh variants. (a) Target-value attention at F16/F20/F24. (b) Fraction of answers with every token correct under value-side interventions on all three late calls; bars show 95% intervals.

Both variants keep G1's single parameter set but refresh K or V at F16. K-refresh allocates more attention to target values at all three late calls (62.2% versus 45.6% at F16). Zeroing values at all non-target positions ("target V only") raises V-refresh from 49% to 82% and K-refresh from 60% to 96%. Content remains in the cache; the difference lies in attending to it, matching Section 2.2's distinction between content being present and being addressable. Refreshing one side creates no new (κ, ν) pair and may disrupt K/V alignment.

D.4 The Addressing Gap with a Single KV Cache

On a UUID set with $n = 400$, we compare native addressing; background-value removal (zero values at non-answer positions, native addressing); a region oracle that restricts late-call attention to the correct UUID span and renormalizes the original weights; and gold addressing that specifies the source token to copy. Intervals use 20,000 paired bootstrap samples.

Architecture	Native	Background V removed	Region oracle	Gold	Dilution	Fine misalignment	Misalignment share (95% CI)
Baseline	80.5	91.8	85.8	92.5	5.25	6.75	56.3% [39.6,75.6]
G2	78.5	85.5	89.2	96.5	10.75	7.25	40.3% [28.2,53.6]
G1	64.2	70.0	93.2	96.8	**29.0**	3.5	10.8% [5.7,16.5]
KV-only	38.0	49.2	49.5	98.8	11.5	**49.25**	81.1% [74.5,87.7]

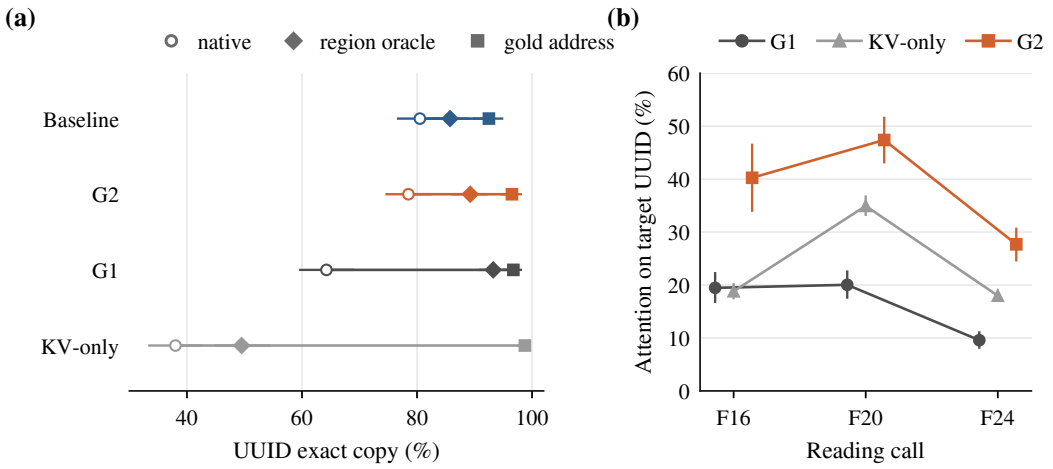

Figure 10: The addressing gap. (a) UUID exact copying under native, region-oracle, and gold addressing; bars show 95% intervals. (b) Attention on the correct UUID by late calls among native failures.

Gold performance is 92–99% for all four architectures: the answer content is retrievable given correct addressing. Dilution is region oracle minus native; fine misalignment is gold minus region oracle (nested, additive, but not independent causes). G1's gap is mainly attention dispersed outside the UUID (region oracle raises it from 64.2% to 93.2%); KV-only's gap lies within the region (region oracle reaches only 49.5%, and recovery requires token-level positions). Panel (b) averages attention over heads and over generation steps up to the first error; native failure counts are 143, 248, and 86 for G1, KV-only, and G2.

E Settings of Table 3

Panel (a) of Table 3 measures free-generation task accuracy after adaptation. For merged G1 we adapt all eighteen GDN layers, so that the residual input writing its K/V also changes; with only the last twelve adapted, G1's K/V would equal group A's by construction. In this setting G1 reaches 22.83% on Multi-key, and the retained function is single-key lookup. Panel (b) decomposes UUID failures of the from-scratch models (Appendix D.4); at the task level, from-scratch G1 and G2 score 33.67% and 37.33% on Multi-key, respectively.

The retained component is consistent with the origin of the shared K/V and with the training signal: merged G1 inherits group A's lookup K/V, and about 87% of answer tokens in the pointer task belong to the UUID. Without distractors (Appendix F.3), G1 shows no deficit, since lookup can then be replaced by fixed-window continuation. Single-cache models store half of G2's full-attention KV.

F Common-Parent Replacement and Chained Retrieval for Sections 4.2–4.3

F.1 Common-Parent Models and Paired Adaptation

We construct a two-group model from a single 0.5B, 24-layer G3 checkpoint. The original `AA BB CC` writes caches at F4/F12/F20; the merged `AA BBBB` replaces F20/F24 with B's complete full-attention layer reading F12 cache. GDN initialization, depth, and all six calls are preserved. This G2 differs from the `AAA BBB` architecture trained from scratch.

All full-attention parameters, embeddings, and final normalization are frozen. Original G3 and each merged model adapt the final twelve GDN layers with matched examples, order, and update budgets. Three independent training pools each contain 256 questions per task across six tasks (1,536 total). Each model receives 512 updates of three questions, using normalized SGD on GDN parameters with FP32 master weights and update rate 10^{-4}. Each task additionally has 4 calibration, 32 development,

and 200 test questions (1,200 test questions). Keys, numbers, UUIDs, and insertion positions are regenerated; no key–value fact is shared across train/test or across pools. We use the fixed 512-step endpoint without test-based selection. Evaluation uses free greedy generation for up to 128 tokens; repeats vary adaptation data and order while sharing one parent.

Task	Original G3	Direct merge	Adapted G3	Adapted G2
Single-key digits / repeated background	100.00	100.00	99.83	99.67
Single-key digits / prose background	100.00	100.00	100.00	100.00
UUID	82.50	76.50	98.83	99.50
Numeric multi-key	32.50	35.00	98.00	97.50
Multi-value (recall)	24.38	25.00	74.46	76.67
Multi-query (recall)	24.12	24.38	62.29	63.46
Six-task mean	60.58	60.15	88.90	89.47

Adapted columns average three repeats. Because all runs share one parent, these results support comparable performance of G2 and G3, not superiority of G2. The 30 merge-damaged questions (21 UUID, 9 multi-key; unused for training or selection) are repaired in all three repeats (30/30); G2 retains 627/630/627 of the questions originally correct in G3, versus 627/630/628 for adapted G3. Eight of the nine multi-key failures after direct merging output another entity's complete value, so repair restores the correct association rather than merely number formatting.

F.2 One-Group Control

G1 uses A's parameters and F4 K/V for all six calls. Direct construction gives means of 0.00% for G1 versus 60.15% for G2, so initial damage is unequal. Allowing all eighteen GDN layers to adapt in every architecture gives:

Model	Repeat 1	Repeat 2	Repeat 3	Mean	UUID mean	Multi-key mean
G1	52.52	50.81	50.38	51.24	0.00	22.83
G2	90.77	90.25	89.62	90.22	97.17	98.00
G3	91.44	91.00	89.92	90.78	99.17	98.00

This supports greater practical adaptability of the two-group construction, not an expressivity limit for every one-group interface.

F.3 Multi-Hop Chained Retrieval

Task. Each context contains sixteen pointer chains $k_0 \to \cdots \to k_4$, and each chain end k_4 holds a random UUID. Records read "`a` ⟨distractors⟩ `points to b.`" and "`k` ⟨distractors⟩ `holds` ⟨UUID⟩ `.`", with 2–10 random short words between identity and relation word. All 80 records are shuffled (about 1,420 tokens). Chain names are single-token words of 5–8 letters and distractors single-token words of 3–4 letters (disjoint sets). Questions start at k_{4-h}, $h \in \{0, \ldots, 4\}$, and ask for the h intermediate words followed by the UUID; the context distribution is identical across h. The distractors instantiate the second case of Proposition A.1, so every hop requires one lookup; the UUID (about 32 tokens) requires extended continuation. By Theorem 1 and Proposition A.2, the answer executes step by step during generation.

Protocol. Starting from the 0.5B G1 and G2 checkpoints in Table 2, we fine-tune all parameters, keeping shared parameters shared: AdamW with fp32 master weights, $\beta = (0.9, 0.95)$, weight decay 0.01, learning rate 10^{-5} with 100 warmup steps and cosine decay, 3,000 steps of eight fresh examples (h uniform), loss on answer tokens only, identical example streams. The test set has 200 questions per hop, shared by both models; evaluation uses cached greedy generation and exact match of the UUID. At learning rate 3×10^{-5}, dev accuracy of Baseline, G1, G2, and G3 all declines after step 500 together with teacher-forced intermediate-word accuracy, so those runs are not used for comparison.

Table G.1: Per-hop chained-retrieval accuracy (% exact UUID, 200 questions per hop; difference paired by question, 95% intervals from 5,000 bootstrap resamples).

	$h = 0$	$h = 1$	$h = 2$	$h = 3$	$h = 4$	Mean
G2	100.0	97.0	98.0	97.5	97.0	97.9
G1	99.5	96.5	95.0	90.5	89.0	94.1
Difference	+0.5	+0.5	+3.0	+7.0	+8.0	+3.8
Interval	$[0.0, 1.5]$	$[-3.0, 4.0]$	$[0.0, 6.0]$	$[2.5, 11.5]$	$[3.0, 13.0]$	$[2.2, 5.5]$

Errors occur almost exclusively on intermediate words: 20 of G2's 21 errors and 56 of G1's 59. Teacher-forced UUID-token accuracy is at least 99.98% for both models, and intermediate-word-token accuracy is 99.0% (G2) and 97.9% (G1). G1's deficit therefore lies in locating the next record across distractors, and it accumulates with the number of lookups.

Control without distractors. When identity and content are adjacent ("`a points to b.`", six-digit number at the chain end), fixed-window continuation reaches the next record. Six architectures trained on this format (mixed-format training, learning rate 3×10^{-5}, 8,000 steps, two seeds) reach 97–100% at every hop, with G1 at 98.8–99.0% and G2 at 99.5–100%. This control also changes content type and protocol and serves only as a reference.

G Fixed K/V Generation and Content Substitution for Section 4.4

Table 6: **Fixed K/V generation and content substitution.** Top: only the nine GDN layers after F12 are adapted; G2's A/B K/V remain fixed, while G3's C K/V may change. Six-task means, ranges over three repeats. Bottom: only the target value is changed at the same adapted G2 checkpoints; complete-new-target generation over 288 questions.

Control	Condition	Result (%)
Fixed K/V generation	G2 (unchanged A/B K/V)	74.86 (74.31–75.42)
Fixed K/V generation	G3 (matched budget)	72.97 (72.31–74.04)
Content substitution	Naturally edited input	76.85
Content substitution	Target-position K/V replacement	75.12
Content substitution	V-only replacement	19.10

G.1 Fixed-Cache Adaptation

The last twelve GDN layers of Appendix F.1 include computation before F12, so B's cache may change. The fixed-cache control trains only the nine GDN layers 13/14/15/17/18/19/21/22/23, freezing all full-attention layers and everything upstream of the A/B writers; G2's A/B K/V are byte-identical before and after adaptation for identical inputs. G3 uses the same nine layers and budget, but its C cache at F20 may change. G2 reaches 74.31/75.42/74.85 (mean 74.86) versus 72.56/74.04/72.31 (mean 72.97) for G3, retaining 615/621/614 versus 616/621/618 of the 630 originally correct questions; all 21 merge-damaged UUID questions recover in every G2 repeat. Multi-key is 60.33% versus 58.33%, but all-target accuracy on multi-value/query stays below 1% for both, so this is relative replacement under a restricted budget, not a solution of multi-target tasks.

G.2 Complete New Values Enter Answers through the Shared Cache

At the three adapted G2 checkpoints, we keep each question's original input, answer-start state, and GDN cache. At the original record positions (value tokens and following period), we replace local K/V with B-cache entries produced by naturally editing the value, only at answer-prediction positions; no donor residual state, GDN cache, or answer prefix is transplanted. The main set has 48 questions per task across six tasks (288); the 21 merge-damaged UUID questions form a separate set. We audit 4,248 outputs.

Task	Naturally edited input	Record V transplant	Matched record K/V transplant
Single-key digits / repeated background	100.00	61.11	100.00
Single-key digits / prose background	100.00	49.31	100.00
UUID	98.61	0.00	97.22
Numeric multi-key	66.67	2.78	62.50
Multi-value: new target	58.33	0.69	56.25
Multi-query: new target	37.50	0.69	34.72
Six-task mean	76.85	19.10	75.12

Matched K/V minus natural editing is -1.74 points (question-paired bootstrap 95% CI $[-3.36, -0.23]$): K/V transplantation suffices to direct complete-new-value generation, though slightly below natural re-encoding. On the 21 recovered UUID questions, matched record K/V gives 21/21 complete new values in each repeat (16/21 before adaptation), whereas V-only gives 0/21; transplanting value tokens without the following period yields correct prefixes but no valid terminated value, while including the period yields 63/63 complete values ending in EOS.

Limits. Transplanting non-target record K/V changes native multi-key correctness from 98/144 to 86/144 (19 failures, 7 repairs), so records are not interchangeable without interference. Target transplantation loses 44/204 and 53/174 other targets on multi-value and multi-query, versus 37/204 and 51/174 under natural editing. Content transfer through the shared cache is established; lossless retention of every downstream function is not.

H Additional Results for Section 5

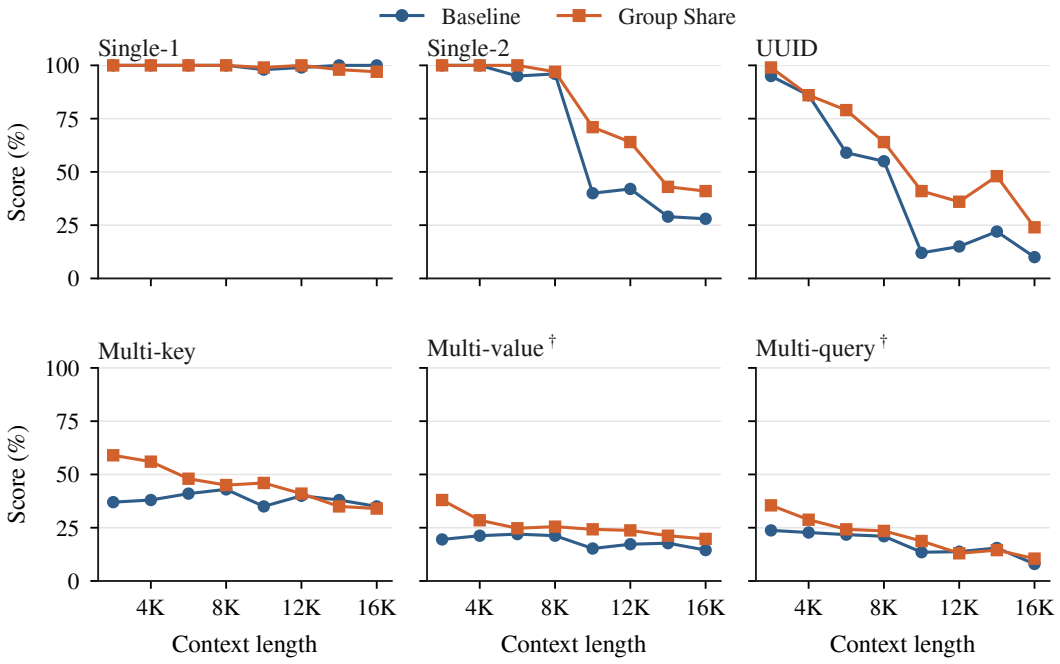

Figure 11: **Six retrieval tasks across context lengths.** Single-1/2, UUID, and Multi-key report exact match; Multi-value and Multi-query, marked †, report complete-target-value recall. Each task, length, and model uses 300 fixed examples.

I MULTI-DEPTH READING AND CROSS-DEPTH REUSE IN QWEN3.5-4B

In pretrained Qwen3.5-4B-Base (32 layers, full attention at 4/8/.../32), we test whether multiple full-attention depths contribute and whether deeper states can read earlier K/V. We use no Group Share checkpoints and do not interpret untrained transplantation as a quality evaluation.

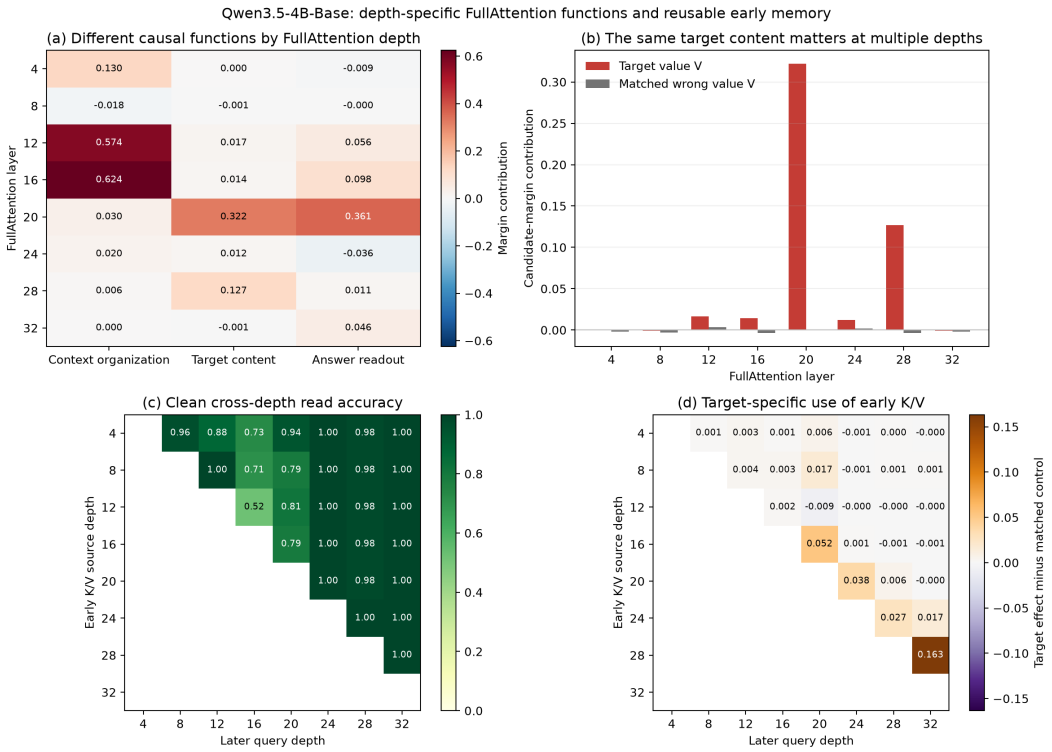

Figure 12: **Full-attention calls in Qwen3.5-4B-Base** (48 multi-key examples). (a) Candidate-margin decrease under the three interventions of Table I.1 (raw correct-V effect before subtracting the control). (b) Correct-V and matched incorrect-V replacement effects. (c) Clean candidate accuracy when the source block reads its own K/V from the target residual state, for all 28 source<target pairs. (d) Additional margin decrease from blocking correct-value minus length-matched incorrect-value tokens in the same pairs.

Multiple depths contribute. Each of 48 multi-key examples contains 2,048 tokens and sixteen key–value pairs; the *candidate margin* is the correct candidate's mean token log-probability minus the highest incorrect score. At each full-attention layer we (i) skip attention at context positions, (ii) replace correct-value V with the unrelated-position mean, minus a matched incorrect-value control, and (iii) skip attention at answer positions; residual paths and FFNs are preserved.

Table I.1: Interventions on Qwen3.5-4B-Base's full-attention layers (mean candidate-margin decrease over 48 examples).

Layer	Skip attention at context positions	Correct V replacement minus control	Skip attention at answer positions
4	0.130	0.002	-0.009
8	-0.018	0.003	-0.000
12	**0.574**	0.013	0.056
16	**0.624**	0.018	0.098
20	0.030	**0.322**	**0.361**
24	0.020	0.011	-0.036
28	0.006	**0.130**	0.011
32	0.000	0.001	0.046

Layers 12/16 contribute through context updates, layer 20 through target content and prediction-time reading, and layer 28 retains a content effect: distinct roles at multiple depths contribute, so reads should be retained.

Deeper states can read earlier K/V. For all 28 source<target pairs, we apply the source block, with its saved K/V, to the target residual state, and separately block correct-value and length-matched incorrect-value tokens; a larger decrease for correct-token blocking indicates use of source-cache answer content at the target depth.

Table I.2: The most consistent cross-depth reads in Qwen3.5-4B-Base.

Source → Target	Unmasked candidate accuracy	Additional decrease from blocking correct tokens	Examples with positive effect
20 → 24	100%	0.038	43/48
24 → 28	100%	0.027	44/48
28 → 32	100%	0.163	45/48
20 → 28	97.9%	0.006	32/48
24 → 32	100%	0.017	34/48

Positive effects recur in the latter half, including two-gap pairs, so reading again need not entail regenerating K/V. Early adjacent pairs (4→8, 8→12, 12→16) average only 0.002 versus 0.070 for late pairs, consistent with independently trained projection spaces never being trained for compatibility. These native-model observations motivate decoupling the number of reads from the number of independent full-attention layers; Sections 4–5 test this decoupling in trained grouped models.