WHITEMATTER: ALL-TO-ALL CROSS-LAYER CONNECTIONS VIA KV SOURCE MIXING

Anonymous authors
Paper under double-blind review

ABSTRACT

When generating text, a Transformer produces representations of past tokens at every layer, but each layer can normally use only representations from the same depth. This restriction prevents the model from fully reusing information it has already computed. We introduce **WhiteMatter**, which allows every layer to draw on past-token representations from any depth. A learned mixer selects the most useful depths for the current context and combines their representations into shared key–value (KV) cache channels. Sharing these channels across layers can reduce the cache size. Given the same number of training tokens, WhiteMatter with a full-size cache performs comparably to a standard Transformer with 50% more layers. With half the KV cache, WhiteMatter outperforms matched standard Transformers at two model scales, up to 1.3B parameters. Cross-layer connections, however, introduce dependencies that slow training and prompt processing. We address this problem with cyclic iteration, which updates interleaved groups of tokens in turn while processing the tokens within each group in parallel. On a reference model trained with exact autoregressive execution, cyclic iteration converges $12.5\times$ faster than standard Jacobi iteration.

1 INTRODUCTION

At each decoding step, a Transformer (Vaswani et al., 2017) processes the current token through every layer before moving to the next token. Yet each layer attends to earlier tokens using only keys and values (KV) produced at the same depth. This restriction makes training and prompt processing easy to parallelize, but prevents layers from reusing representations already computed at other depths. The tradeoff matters because inference can cost more than training (OpenAI, 2020), particularly for agents and long reasoning traces (Saxena et al., 2025; Yuan et al., 2026). It is therefore important to relax this restriction while retaining the practical benefits of the standard Transformer.

Deep-to-shallow feedback offers one way to reuse more of this computation. Feedback Transformer (Fan et al., 2021) combines all layer states for each past token into one weighted sum, then projects that summary into KV shared by every attending layer. LCKV (Wu & Tu, 2024) instead

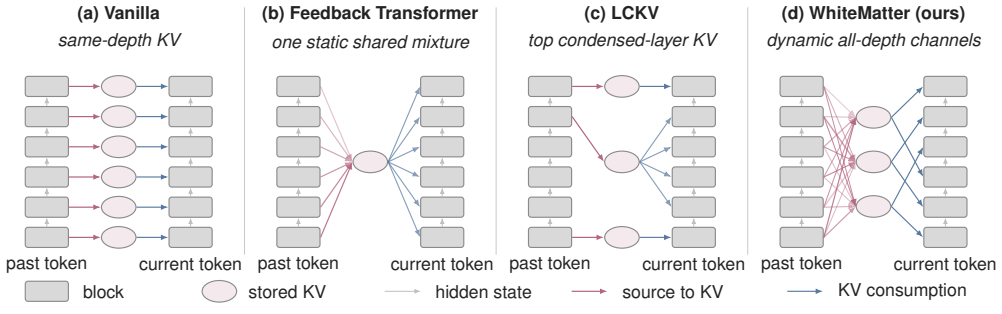

Figure 1: **KV production and consumption across layers.** **(a)** Vanilla: each layer reads KV produced at the same depth. **(b)** Feedback Transformer: all layers read KV produced from the same weighted sum of source depths. **(c)** LCKV: warmup layers retain same-depth KV, while condensed layers read KV produced from the top layer in the block. **(d)** WhiteMatter: a router forms k channels from all source depths, and each layer reads one of the channels.

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.

builds shared KV from the deepest state, which has accumulated information from earlier layers through residual connections. LCKV performs better when some "warmup" layers retain their own same-depth KV, whereas applying feedback KV to every layer substantially worsens perplexity (Wu & Tu, 2024, Figure 8). This raises a central question: why does broader feedback become less effective when it is used everywhere?

One likely reason is the shared bottleneck in both methods: they compress the available source states into a single layer-width representation used by every target layer. Such a summary must preserve every feature that any layer may need. In practice, intermediate states can remain useful even after deeper layers have transformed them, and our ablations show that simply enlarging the KV cache is not enough. We therefore hypothesize that different target layers need access to different source depths. LCKV's warmup layers partly restore intermediate information, but only through the same-depth connections of a standard Transformer.

This need for selective, target-specific connectivity has a loose analogue in the brain. White matter forms a dense network of axons between regions, but communication is not uniform: regions have specialized pathways, and ongoing neural activity modulates which connections exert the greatest influence (Appendix A; Markov et al., 2014; Fries, 2015). Motivated by this combination of broad reach and context-dependent selection, we propose **WhiteMatter** to dynamically channel information across all depths (see Figure 1). It allows every layer to draw on past-token representations from any depth. For each target layer and past token, a context-dependent router mixes the source depths into a specialized representation, which is then projected into a KV channel. A pool of k channels can be shared across layers, trading finer connection specialization for a smaller KV cache.

These feedback connections are straightforward during decoding, when deeper states of past tokens have already been computed, but their dependencies complicate parallel training and prefill. Feedback Transformer processes tokens sequentially, limiting GPU parallelism and scalability (Fan et al., 2021; Wu & Tu, 2024). LCKV uses Jacobi iteration instead: each pass processes all tokens in parallel but can read only the KV produced by the preceding pass (Wu & Tu, 2024). Because updates move forward only between passes, convergence requires many repeated evaluations of the full sequence. We introduce cyclic Gauss–Seidel iteration (Figure 3), which processes strided token groups in order while evaluating the tokens within each group in parallel. Later groups can use updates from earlier groups in the same pass, allowing information to propagate farther with each full-sequence evaluation.

We evaluate both the quality benefit and execution cost of this design. Under matched training tokens, full-cache WhiteMatter achieved 7.7% lower perplexity than vanilla with the same number of layers and performed comparably to vanilla with 50% more layers. Half-cache WhiteMatter lowers held-out perplexity by 5.9% and 4.3% relative to matched vanilla models, up to approximately 1.3B parameters. The 1.3B WhiteMatter model has decoding throughput comparable to vanilla's, with 39.4% lower peak device memory and higher prefill cost. Controlled studies show that cyclic iteration converges faster than Jacobi, and that training closer to convergence improves quality and stability under further refinement.

2 Related Work

Deep-to-shallow feedback connections. Feedback Transformer pools past-token layer states into a shared memory (Fan et al., 2021). LCKV uses top-layer KV for condensed layers and same-depth KV for warm-up layers, with Jacobi iteration for parallel training (Wu & Tu, 2024). T2MLR injects a prior token's middle-layer state into an earlier layer's residual stream (Cai et al., 2026), while Recurrent Transformer constructs each layer's KV from that layer's past outputs (Oncescu et al., 2026). Concurrent with our work, Latent Recurrent Transformer (LRT) (Huang et al., 2026) proposes an interleaved training schedule with a single refinement sweep but does not study how multiple interleaved passes affect convergence. These methods restrict feedback to prescribed source layers or a single shared mixture, whereas WhiteMatter produces content-dependent mixtures specialized for different layers.

Latent reasoning and looping. Staircase attention reprocesses past tokens alongside new tokens (Ju et al., 2022). Coconut and the PonderLM family introduce latent computation through recycled representations or additional sequence positions (Hao et al., 2025; Zeng et al., 2026; Song et al.,

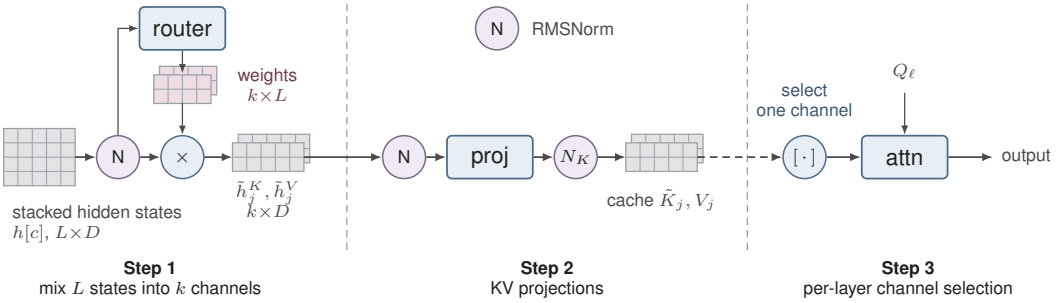

Figure 2: **The cross-layer KV pool for one token position i.** Dashed dividers separate the three steps of §3.1. In Step 1 a data-dependent router mixes the L per-layer states into k shared channels. In Step 2 the resulting channels undergo KV projection; K normalization and RoPE are then applied to the keys before cache storage. In Step 3 each query-side layer reads one channel; the dashed arrow marks the cache boundary, as the stored channels are read while processing a later token. The key and value branches are processed independently.

2026; Zeng et al., 2025; Li et al., 2026). These methods add intermittent feedback connections while increasing computation in decoding.

Feedforward cross-layer aggregation. Several architectures improve information flow through depth by modifying residual connections, using learned aggregation of earlier-layer representations (Pagliardini et al., 2024; Menghani et al., 2025; Kimi Team, 2026; Luo et al., 2026) or mixing of multiple residual streams (Zhu et al., 2025; Xie et al., 2025). DeepCrossAttention and MUDDFormer form separate mixtures of earlier representations for attention inputs (Heddes et al., 2025; Xiao et al., 2025). Value-residual methods reuse first-layer values (Zhou et al., 2024; Gunasekaran et al., 2026), while KV sharing and reconstruction methods apply cross-layer reuse to reduce cache storage (Brandon et al., 2024; Zuhri et al., 2024; Sun et al., 2024; Wu et al., 2025; Filippova et al., 2026). FusedKV stores KV only in the lower half of the layers and reconstructs each upper layer's keys and values through separate learned, channel-wise mixtures of the first and middle layers' caches (Lin et al., 2026). These methods demonstrate the benefit of reusing representations from earlier layers.

3 METHOD

For a Transformer decoder with L layers and width D, let T be the sequence length and i, ℓ, and j index tokens, layers, and channels. WhiteMatter replaces per-layer KV projections with a *cross-layer KV pool*: a router mixes all L states into $k \leq L$ channels, each with its own KV projection. Each layer reads one channel, so the KV cache size is k/L of vanilla's.

3.1 CROSS-LAYER KV POOL

Mixing L states into k channels. Let $h_\ell[i] \in \mathbb{R}^D$ be the hidden state entering layer ℓ at token i. At each position i, the pool combines the L source states into k channels using dynamic mixing weights, computed independently for the key and value branches. We describe the key branch; the value branch is identical with its own parameters.

Each source state is first RMS-normalized, giving $\hat{h}_\ell^K[i]$. This pre-mix norm puts the L layers on a common scale and keeps their magnitudes from growing as they recur through the feedback loop.

The router reads normalized states from every pth source layer to reduce its parameter count, while producing mixing weights over all L layers. Let $\xi^K[i]$ concatenate the selected states. The mixing weights and resulting channel are

$$\alpha_j^K[i] = W_j^{\alpha K}\, \xi^K[i] + b_j^{\alpha K}, \qquad \tilde{h}_j^K[i] = \sum_{\ell=0}^{L-1} \alpha_j^K[i][\ell]\, \hat{h}_\ell^K[i].$$

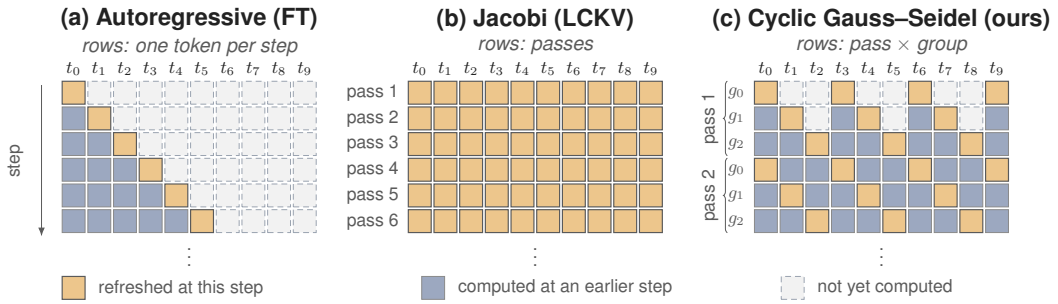

Figure 3: **Schedules for resolving the feedback connections.** Rows are computation steps and columns are tokens; each cell is shaded according to when its KV source was last updated. **(a)** Autoregressive: Feedback Transformer processes each token sequentially. **(b)** Jacobi: LCKV processes all tokens in parallel in multiple passes. **(c)** Cyclic: strided groups run in order, so later groups read earlier groups' updates within a pass.

The weights are signed and can therefore express differences among layer representations. The value branch uses the same construction with its own norm, router $W_j^{\alpha V}, b_j^{\alpha V}$, and weights $\alpha_j^V[i]$ applied to $\hat{h}_\ell^V[i]$.

KV projections. A second RMSNorm places the mixed channels at a common scale before they are projected into keys and values:

$$K_j[i] = W_j^K \,\mathrm{RMSNorm}_j^K(\tilde{h}_j^K[i]), \qquad V_j[i] = W_j^V \,\mathrm{RMSNorm}_j^V(\tilde{h}_j^V[i]),$$

Per-channel key normalization and RoPE are applied before caching the keys; values are cached directly. The pool constructs each token's channels independently from its layer states, so KV can be refreshed at selected positions without recomputing the remaining cache.

Per-layer channel selection. Layer ℓ reads channel $j = \ell \bmod k$. With $k = L$, each layer receives a distinct mixture; smaller k shares mixtures across layers and reduces the KV cache to k/L of vanilla's size. Reading all channels would require each layer to stream all k key and value channels from HBM. The fixed selection preserves one channel read per layer. Appendix B specifies router parameterization, initialization, and cache indexing.

3.2 Autoregressive decoding

WhiteMatter's KV channels depend on states from all depths. Reading the current token's channels would therefore create a dependency cycle between shallow and deep layers. We use *strictly causal attention*: token i attends only to positions $s < i$. After its layer-stack evaluation, we construct its channels and append them to the cache for subsequent tokens. Decoding requires one layer-stack evaluation and one pool evaluation per token; a learned boundary token supplies the initial KV entry.

3.3 Parallel training and prefill

Let $\mathrm{KV}[i]$ denote the key and value channels at position i. Let $\mathrm{Model}(X; \mathrm{KV}_{\mathrm{in}})$ denote one complete model pass that reads $\mathrm{KV}_{\mathrm{in}}$ and returns the newly constructed cache $\mathrm{KV}_{\mathrm{out}}$. Because of the strict causality introduced in §3.2, the KV produced by exact autoregressive execution for an input token sequence $X = (x[0], \ldots, x[T-1])$ satisfies

$$\mathrm{KV} = \mathrm{Model}(X; \mathrm{KV})\,.$$

The cache can therefore be viewed as a fixed point and solved by iteration, thereby reducing the number of sequential operations needed in training and prefill, as shown in Figure 3.

Jacobi iteration. Wu & Tu (2024) approximate the fixed point with n token-parallel passes, analogous to the Jacobi method for solving linear systems. We first initialize $\mathrm{KV}^{(0)}$ from each token's embedding, treating the embedding as its state at every source layer. For $t = 1, \ldots, n$, we update

$$\mathrm{KV}^{(t)} = \mathrm{Model}\Big(X; \mathrm{KV}^{(t-1)}\Big)\,.$$

All T tokens are evaluated in parallel using the previous pass's cache. This maximizes token parallelism, but updates cannot propagate between tokens until the next pass, so convergence can require many full-sequence evaluations.

Cyclic Gauss–Seidel iteration. To propagate updates within a pass while retaining token parallelism, we divide token positions into g groups and process groups in sequence; tokens within each group are evaluated in parallel. We refresh KV after each group so that later groups can read its updates, analogous to Gauss–Seidel iteration. Since attention often concentrates on nearby tokens, we use a cyclic group assignment $\mathcal{G}_q = \{i : i \bmod g = q\}$, evaluated in order $q = 0, \ldots, g-1$, to favor propagation between neighboring positions. For example, each token in group $q > 0$ can read its immediate predecessor's current-pass KV. Entries from groups not yet processed retain their previous-pass values, and all reads obey the same strictly causal $s < i$ mask. Algorithm 1 gives the complete schedule. We implement a cyclic attention kernel by adapting FlashAttention-style tiling (Dao, 2024) to the strided causal mask.

Each pass contains g sequential group evaluations, with approximately T/g tokens evaluated in parallel per group. Increasing g allows updates to propagate through more groups within a pass, but reduces the parallel work available in each group. Minimizing execution time therefore requires balancing the number of passes required and the cost of each pass.

Algorithm 1 Cyclic iteration

Input: Token embeddings X, number of passes n, number of groups g

Initialize KV from token embeddings
for $t = 1, \ldots, n$ **do**
  for $q = 0, \ldots, g-1$ **do**
    $\mathcal{G}_q \leftarrow \{i : 0 \le i < T,\ i \bmod g = q\}$
    Evaluate tokens $\mathcal{G}_q$ in parallel using current KV
    Collect their layer inputs H and final outputs $Y[\mathcal{G}_q]$
    $\mathrm{KV}[\mathcal{G}_q] \leftarrow \mathrm{Pool}(H)$
  end for
end for
return Y, KV

Truncated backpropagation. During training, increasing the number of refinement passes also increases the cost of backpropagation. We follow Wu & Tu (2024) in carrying gradients only through the last $n_g \le n$ passes; earlier passes run without gradients and serve to approach the fixed point.

4 Experiments

4.1 Experimental setup

Models and baselines. We evaluated Qwen3-based decoders (Yang et al., 2025) at two scales. At D=512, we compared 16-layer WhiteMatter with full (k=16) and half (k=8) KV caches against vanilla, LCKV (Wu & Tu, 2024), and FusedKV (Lin et al., 2026). We also trained a 24-layer vanilla model to compare against greater depth. At D=1792, we compared 28-layer vanilla and half-cache WhiteMatter (k=14), with 1.351B and 1.326B parameters, respectively. WhiteMatter used g=8 cyclic groups and router stride p=2.

LCKV used $w \in \{4, 7\}$ same-depth warmup layers around a condensed block sharing one KV source. The bottom/condensed/top layer counts were $2/12/2$ and $3/9/4$, giving $5/16$ and $1/2$ of vanilla's cache, respectively; w=7 matches half-cache WhiteMatter. FusedKV retained KV in the first eight layers and used layers 0 and 7 as sources for the upper eight layers. Appendix C gives the remaining architecture settings.

Data and training. We trained from scratch on shuffled FineWeb-Edu (Penedo et al., 2024) with the Qwen3 tokenizer, packing length-2048 sequences with EOS separators and within-document attention. The D=512 and D=1792 suites used matched budgets of 8B and 10B tokens and effective batches of 128 and 32, respectively. WhiteMatter used one no-gradient cyclic pass followed by two gradient-carrying passes. LCKV used seven no-gradient Jacobi passes followed by two gradient-carrying passes (Wu & Tu, 2024); vanilla and FusedKV used one pass. We used Muon (Jordan et al., 2024) for two-dimensional weight matrices and AdamW for the remaining parameters. Appendix C gives optimizer and batching details.

Quality evaluation. We measured perplexity on 5,000 held-out sequences and evaluated zero-shot benchmarks with lm-evaluation-harness (Gao et al., 2023). All evaluations use final checkpoints and full evaluation splits. WhiteMatter and LCKV use three cyclic and nine Jacobi passes, respectively, for

Table 1: **Zero-shot evaluation at D=512 with 8B training tokens.** Both full- and half-cache WhiteMatter models achieved lower perplexity and higher average accuracy than all other models, including the 24L vanilla model, which has 50% more layers.

Model	Wiki. PPL ↓	LMB. PPL ↓	LMB. acc. ↑	BLiMP acc. ↑	PIQA acc. ↑	Hella. acc. n ↑	ARC-E acc. ↑	ARC-C acc. n ↑	Wino. acc. ↑	OBQA acc. n ↑	SciQ acc. ↑	ReC. EM ↑	SQD acc. ↑	Avg. acc. ↑
Vanilla 16L	49.34	127.47	24.43	77.23	62.73	31.67	52.15	25.00	50.04	29.00	78.50	58.50	30.03	47.21
FusedKV	48.35	92.33	27.13	76.64	**63.71**	32.25	50.76	25.00	52.01	30.20	80.30	58.96	34.82	48.34
LCKV w=4	48.81	107.52	26.33	76.17	62.73	32.52	51.43	23.72	50.91	**31.20**	76.60	58.04	**45.48**	48.65
LCKV w=7	49.02	102.97	25.87	75.73	61.97	32.40	52.44	**25.94**	**52.41**	30.00	76.50	58.34	44.00	48.69
WhiteMatter k=8	44.40	71.58	29.65	78.83	63.22	33.61	52.65	25.26	51.38	29.60	80.50	60.24	36.70	49.24
WhiteMatter k=16	**43.28**	**60.73**	**30.60**	**79.88**	63.33	**33.80**	**52.69**	25.43	51.38	29.40	**83.00**	**61.52**	37.80	**49.89**
Vanilla 24L	44.71	97.40	26.90	77.73	63.49	33.21	53.16	26.37	50.99	31.80	78.20	59.40	32.54	48.52

Table 2: **Zero-shot evaluation at D=1792 with 10B training tokens.** At 1.3B total parameters, WhiteMatter also consistently outperforms vanilla while using 50% less KV cache.

Model	Wiki. PPL ↓	LMB. PPL ↓	LMB. acc. ↑	BLiMP acc. ↑	PIQA acc. ↑	Hella. acc. n ↑	ARC-E acc. ↑	ARC-C acc. n ↑	Wino. acc. ↑	OBQA acc. n ↑	SciQ acc. ↑	ReC. EM ↑	SQD acc. ↑	Avg. acc. ↑
Vanilla 28L	26.61	23.84	39.65	80.83	**70.46**	49.80	65.74	32.59	55.49	37.60	85.90	70.25	44.00	57.48
WhiteMatter k=14	**24.67**	**18.42**	**43.80**	**81.38**	69.53	**52.72**	**67.21**	**36.01**	**57.38**	**37.80**	**88.70**	**72.75**	**44.34**	**59.24**

In both tables, SQD is SQuAD cloze completion scored by answer containment. Avg. is the unweighted mean of the 11 non-perplexity scores. acc. n denotes length-normalized accuracy and EM denotes exact match.

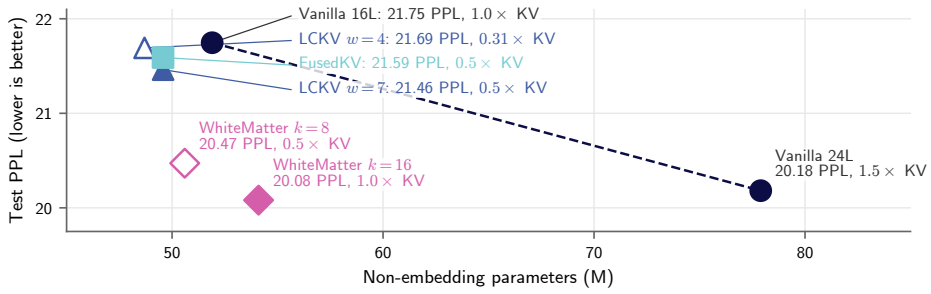

Figure 4: **Language-modeling quality versus model size after 8B training tokens.** Lower perplexity is better; labels indicate KV-cache size relative to vanilla 16L. Full-cache WhiteMatter performs comparably to the deeper vanilla model, while half-cache WhiteMatter improves over same-depth vanilla.

likelihood evaluation and generation prefill; generation then uses cached autoregressive decoding. We also evaluate the perplexity under autoregressive decoding for WhiteMatter and LCKV. Appendix H gives task and scoring details.

Runtime evaluation. We measured throughput and memory for the 1.3B architectures at batch 64 on an RTX A6000 with compiled BF16 execution. Prefill processes 2048 prompt tokens; decoding generates 128 tokens after this prefix. WhiteMatter uses the evaluation schedule above. LCKV uses nine Jacobi passes with a 6/15/7-layer sandwich, and Feedback Transformer processes prompt tokens autoregressively. See Appendix D for measurement details.

4.2 Language Modeling and Downstream Performance

Language-modeling quality. Half-cache WhiteMatter reduces held-out perplexity by 5.9% at the smaller scale and 4.3% at approximately 1.3B parameters, relative to matched vanilla models. At the smaller scale, it also outperforms equal-cache LCKV and FusedKV, while full-cache WhiteMatter achieves perplexity comparable to the deeper vanilla model with fewer non-embedding parameters (Figure 4). For all evaluated WhiteMatter and LCKV models, autoregressive perplexity differs by less than 1% from that obtained with fixed-point iteration.

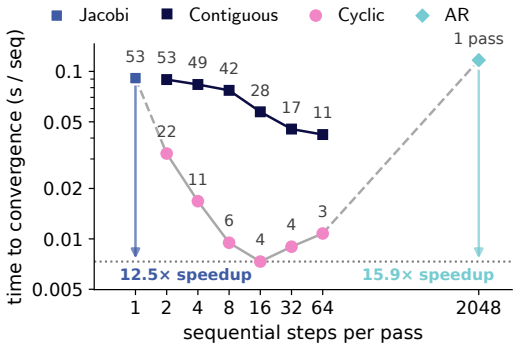

Figure 5: **Prefill runtime at matched PPL.** Labels show pass counts; AR denotes autoregressive execution. Increasing the number of sequential steps per pass reduces the passes needed but increases the cost of each pass. Cyclic iteration balances these effects, achieving the lowest runtime and a 12.5× speedup over Jacobi.

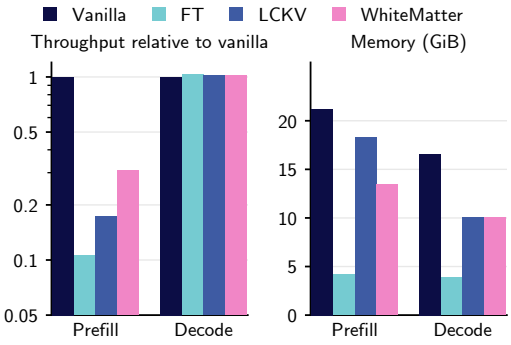

Figure 6: **Inference throughput and peak device memory.** Throughput (log scale) is relative to vanilla. FT denotes Feedback Transformer. WhiteMatter achieves decoding throughput comparable to vanilla's with lower memory use. Its prefill is slower than vanilla's but faster than that of the other feedback architectures.

Zero-shot performance. Half-cache WhiteMatter improves the average downstream score over matched vanilla models at both scales (Tables 1 and 2). At the smaller scale, both half- and full-cache WhiteMatter outperform all baselines on average, including the vanilla model with 50% more layers, with full-cache WhiteMatter achieving the highest average score.

4.3 Prefill convergence

We evaluated prefill convergence on a 4-layer model trained with exact autoregressive execution for 78.6M tokens. For each schedule, we selected the first pass whose perplexity was at most 1% above the exact autoregressive reference. Pass counts were selected in FP32; Figure 5 reports compiled BF16 decoder runtime at those counts, using length-2048 sequences and batch 32. Appendix F.1 gives more detail.

Cyclic g=16 reaches the target in four passes and takes 7.32 ms/sequence, compared with 53 passes and 91.20 ms/sequence for Jacobi, a 12.5× speedup. More groups reduce the passes needed but increase the cost of each pass, so the largest group count does not give the lowest runtime.

We also evaluated dividing each pass into contiguous chunks processed from left to right. Cyclic groups require fewer passes and less time at every tested group count, supporting their use for faster propagation within a pass.

4.4 Prefill and decoding efficiency

At batch size 64, WhiteMatter prefill achieves 31% of vanilla's throughput, 1.78× LCKV's, and 2.92× Feedback Transformer's (Figure 6). Its peak device memory is 13.44 GiB, compared with vanilla's 21.21 GiB. All four architectures decode at approximately 2,500 tokens/s. WhiteMatter uses 10.05 GiB of peak device memory versus vanilla's 16.59 GiB, a 39.4% reduction. LCKV uses a similar 10.01 GiB, while Feedback Transformer uses 3.86 GiB.

5 Analysis

For the schedule and connectivity experiments, we used the 16-layer, D=512 model and the training setup in §4.1, reducing the budget to 20,000 steps at global batch 8 (327.7M tokens).

Iteration schedules and convergence. We study how training closer to convergence affects model quality. With k=8, we varied gradient passes $n_g \in \{1, 2\}$, no-gradient passes $n_{\text{no-grad}} \in \{1, 2, 4\}$, and schedules $\{\text{Jacobi}, C_4, C_8, C_{16}\}$ over two seeds. Figure 7a reports the best perplexity over passes 1–32 under each training schedule, approximate autoregressive perplexity after $32C_{16}$ passes, and

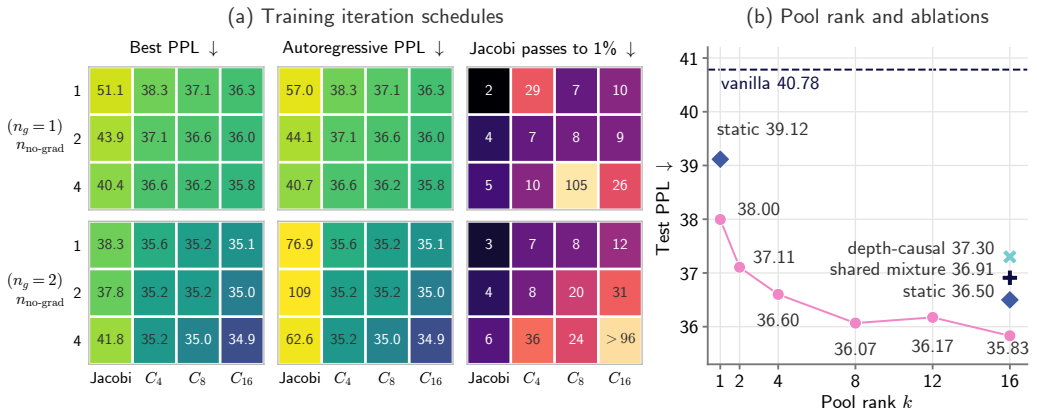

Figure 7: **(a) Training schedules.** Training with more refinement generally improves perplexity and reduces degradation under further iteration, although approaching the best quality can require more inference passes. Row groups vary gradient-carrying passes, inner rows vary no-gradient passes, and columns compare Jacobi with cyclic schedules (C_g: g groups). **(b) Channel count and ablations.** More KV channels generally improve quality, but increasing cache capacity alone does not recover the benefit of specialized source mixtures. Content-dependent routing and deep-to-shallow connections also improve performance.

the Jacobi passes needed to reach within 1% of the best perplexity. The strongest schedule lowers perplexity by 32% relative to the weakest. More passes and cyclic groups improve quality with diminishing returns. Models trained farther from the fixed point degrade under further iteration; those trained closer remain stable after convergence but can require more Jacobi passes to approach their best perplexity.

Computational cost. At D=512, WhiteMatter uses 0.99–$1.03\times$ vanilla's decoding FLOPs, but 2.32–$2.50\times$ its training FLOPs and 3.05–$3.30\times$ its three-pass prefill FLOPs, excluding the LM head (Appendix E). Truncated backpropagation limits training cost, while wall time also depends on parallelism and memory traffic.

We further examine how channel count, distinct source mixtures, content-dependent routing, and deep-to-shallow access contribute to model quality.

Channel count. We trained models with $k \in \{1, 2, 4, 8, 12, 16\}$ (Figure 7b). More channels generally improve quality, most strongly from k=1 to k=2.

Specialized mixtures. We replace the distinct source mixtures with one shared mixture while retaining k=16 independent KV projections. This model performs worse than WhiteMatter with k=4, despite using $4\times$ the cache.

Dynamic routing. Replacing the dynamic router with static learnable weights raises perplexity by 3.0% at k=1 and 1.9% at k=16.

Deep-to-shallow feedback. Restricting layer ℓ to KV mixtures of states $0, \ldots, \ell$ removes feedback and iteration. This full-cache model improves over vanilla, but its perplexity is 4.1% higher than full-cache WhiteMatter, supporting the benefit of deep-to-shallow reuse.

6 Conclusion

We introduced WhiteMatter, which gives Transformer layers access to past-token representations from all depths through content-dependent KV source mixing. Distinct mixtures enable specialized connections, while channel sharing reduces KV-cache storage. Under matched training-token budgets, half-cache WhiteMatter improved perplexity and average downstream performance over vanilla at both evaluated scales, up to 1.3B parameters; at the smaller scale, full-cache WhiteMatter performs comparably to vanilla with 50% more layers. Ablations supported distinct mixtures, dynamic routing, and deep-to-shallow feedback. We also introduced cyclic Gauss–Seidel iteration to accelerate training

and prefill, reaching the target perplexity $12.5\times$ faster than Jacobi on the autoregressively trained reference model. At 1.3B parameters, WhiteMatter matched vanilla decoding throughput with 39.4% lower peak device memory, although iterative training and prefill remained more expensive. Overall, specialized reuse of past-token states improves quality while reducing cache storage; further reducing training and prefill costs remains future work.

LIMITATIONS

Despite speedups relative to autoregressive computation and Jacobi iteration, iterative training and prefill remain more expensive than those of standard Transformers. Our main quality results cover up to 1.3B parameters trained for 10B tokens; effectiveness on larger models trained with more data requires further investigation. Full-cache and alternative KV-sharing baselines are evaluated only at the smaller scale.

AI USE STATEMENT

We used generative AI tools to implement methods.

We did not use generative AI tools to help develop theoretical models or conceptual frameworks, formulate mathematical claims, propose or refine hypotheses, design or provide feedback on research methodology or experiments, support qualitative and thematic data analysis, or interpret results. The following uses were not applicable to this work: generating synthetic data sets; providing critical ingredients for proving mathematical claims; assisting in the writing of proofs; assisting with translation; and cleaning and reformatting datasets.

We also used generative AI tools to create or modify scientific figures or images, create or edit software code, draft parts of the paper, summarize or analyse existing literature, source or search for information, edit the paper to improve readability, identify relevant literature, format references, and suggest a structure for the paper.

We reviewed all AI-assisted work. We performed extensive manual literature surveys alongside LLM-assisted ones. LLM-generated code was verified and tested for correctness. The manuscript was manually reviewed and edited in multiple revisions. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff, 2024.

William Brandon, Mayank Mishra, Aniruddha Nrusimha, Rameswar Panda, and Jonathan Ragan-Kelley. Reducing transformer key-value cache size with cross-layer attention. *arXiv preprint arXiv:2405.12981*, 2024.

Ziyang Cai, Xingyu Zhu, Yihe Dong, Yinghui He, and Sanjeev Arora. T^2MLR: Transformer with temporal middle-layer recurrence. *arXiv preprint arXiv:2607.15178*, 2026.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2307.08691.

David C. Van Essen, Stephen M. Smith, Deanna M. Barch, Timothy E. J. Behrens, Essa Yacoub, Kamil Ugurbil, and WU-Minn HCP Consortium. The WU-Minn human connectome project: An overview. *NeuroImage*, 80:62–79, 2013. doi: 10.1016/j.neuroimage.2013.05.041.

Angela Fan, Thibaut Lavril, Edouard Grave, Armand Joulin, and Sainbayar Sukhbaatar. Addressing some limitations of transformers with feedback memory. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=OCm0rwa1lx1.

Anastasiia Filippova, David Grangier, Marco Cuturi, and João Monteiro. Stochastic KV routing: Enabling adaptive depth-wise cache sharing. *arXiv preprint arXiv:2604.22782*, 2026.

Pascal Fries. Rhythms for cognition: Communication through coherence. *Neuron*, 88(1):220–235, 2015. doi: 10.1016/j.neuron.2015.09.034.

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. A framework for few-shot language model evaluation, 2023. URL https://zenodo.org/records/10256836.

Skye Gunasekaran, Téa Wright, Rui-Jie Zhu, and Jason Eshraghian. Transformers with selective access to early representations. *arXiv preprint arXiv:2605.03953*, 2026.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. In *Conference on Language Modeling*, 2025. URL https://arxiv.org/abs/2412.06769.

Mike Heddes, Adel Javanmard, Kyriakos Axiotis, Gang Fu, MohammadHossein Bateni, and Vahab Mirrokni. DeepCrossAttention: Supercharging transformer residual connections. *arXiv preprint arXiv:2502.06785*, 2025.

Zeyi Huang, Xuehai He, Liliang Ren, Yiping Wang, Baolin Peng, Hao Cheng, Shuohang Wang, Pengcheng He, Jianfeng Gao, Yong Jae Lee, and Yelong Shen. Latent recurrent transformer: Architecture exploration, training strategies, and scaling behavior. *arXiv preprint arXiv:2605.26797*, 2026. URL https://arxiv.org/abs/2605.26797.

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

Da Ju, Stephen Roller, Sainbayar Sukhbaatar, and Jason Weston. Staircase attention for recurrent processing of sequences. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://openreview.net/forum?id=NiCJDYpKaBj.

Kimi Team. Attention residuals. *arXiv preprint arXiv:2603.15031*, 2026.

He Li, Feichen Song, Boyi Zeng, Shixiang Song, Zhiqin John Xu, Ziwei He, and Zhouhan Lin. PonderLM-3: Adaptive token-wise pondering with differentiable masking. *arXiv preprint arXiv:2603.02023*, 2026.

Hongzhan Lin, Zhiqi Bai, Xinmiao Zhang, Sen Yang, Xiang Li, Siran Yang, Yunlong Xu, Jiaheng Liu, Yongchi Zhao, Jiamang Wang, Yuchi Xu, Wenbo Su, and Bo Zheng. Reconstructing KV caches with cross-layer fusion for enhanced transformers. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2512.03870.

Cheng Luo, Zefan Cai, and Junjie Hu. Delta attention residuals. *arXiv preprint arXiv:2605.18855*, 2026.

Nikola T. Markov, M. M. Ercsey-Ravasz, A. R. Ribeiro Gomes, C. Lamy, et al. A weighted and directed interareal connectivity matrix for the macaque cerebral cortex. *Cerebral Cortex*, 24(1): 17–36, 2014. doi: 10.1093/cercor/bhs270.

Gaurav Menghani, Ravi Kumar, and Sanjiv Kumar. LAuReL: Learned augmented residual layer. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 43826–43836. PMLR, 2025. URL https://proceedings.mlr.press/v267/menghani25a.html.

Costin-Andrei Oncescu, Depen Morwani, Samy Jelassi, Alexandru Meterez, Mujin Kwun, and Sham Kakade. The recurrent transformer: Greater effective depth and efficient decoding. *arXiv preprint arXiv:2604.21215*, 2026.

OpenAI. AI and efficiency. OpenAI, May 2020. URL https://openai.com/index/ai-and-efficiency/.

Matteo Pagliardini, Amirkeivan Mohtashami, François Fleuret, and Martin Jaggi. DenseFormer: Enhancing information flow in transformers via depth weighted averaging. In *International Conference on Machine Learning (ICML)*, 2024. URL https://arxiv.org/abs/2402.02622.

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 (NeurIPS) Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2406.17557.

Anish Saxena, Po-An Tsai, Hritvik Taneja, Aamer Jaleel, and Moinuddin Qureshi. Utility-driven speculative decoding for mixture-of-experts. *arXiv preprint arXiv:2506.20675*, 2025. URL https://arxiv.org/abs/2506.20675.

Shixiang Song, He Li, Zitong Wang, Boyi Zeng, Feichen Song, Yixuan Wang, Zhiqin John Xu, Ziwei He, et al. AdaPonderLM: Gated pondering language models with token-wise adaptive depth. *arXiv preprint arXiv:2603.01914*, 2026.

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper/7181-attention-is-all-you-need.

Haoyi Wu and Kewei Tu. Layer-condensed KV cache for efficient inference of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024. URL https://arxiv.org/abs/2405.10637.

You Wu, Haoyi Wu, and Kewei Tu. A systematic study of cross-layer KV sharing for efficient LLM inference. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pp. 396–403. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-short.34. URL https://aclanthology.org/2025.naacl-short.34/.

Da Xiao, Qingye Meng, Shengping Li, and Xingyuan Yuan. MUDDFormer: Breaking residual bottlenecks in transformers via multiway dynamic dense connections. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 68440–68458. PMLR, 2025. URL https://proceedings.mlr.press/v267/xiao25d.html.

Zhenda Xie, Yixuan Wei, Huanqi Cao, et al. mHC: Manifold-constrained hyper-connections. *arXiv preprint arXiv:2512.24880*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Fang-Cheng Yeh. DSI Studio: An integrated tractography platform and fiber data hub for accelerating brain research. *Nature Methods*, 22:1617–1619, 2025. doi: 10.1038/s41592-025-02762-8.

Fang-Cheng Yeh, Sandip Panesar, David Fernandes, Antonio Meola, Masanori Yoshino, Juan C. Fernandez-Miranda, Jean M. Vettel, and Timothy Verstynen. Population-averaged atlas of the macroscale human structural connectome and its network topology. *NeuroImage*, 178:57–68, 2018. doi: 10.1016/j.neuroimage.2018.05.027.

Yichao Yuan, Ankita Nayak, Souvik Kundu, and Nishil Talati. Agentic AI workload characteristics. *arXiv preprint arXiv:2605.26297*, 2026. URL https://arxiv.org/abs/2605.26297.

Boyi Zeng, He Li, Shixiang Song, Yixuan Wang, Zitong Wang, Ziwei He, Xinbing Wang, and Zhouhan Lin. PonderLM-2: Pretraining LLM with latent thoughts in continuous space. *arXiv preprint arXiv:2509.23184*, 2025.

Boyi Zeng, Shixiang Song, Siyuan Huang, Yixuan Wang, He Li, Ziwei He, Xinbing Wang, Zhiyu Li, and Zhouhan Lin. PonderLM: Pretraining language models to ponder in continuous space. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=UrM4MNRYZm.

Zhanchao Zhou, Tianyi Wu, Zhiyun Jiang, Fares Obeid, and Zhenzhong Lan. Value residual learning. *arXiv preprint arXiv:2410.17897*, 2024.

Defa Zhu, Hongzhi Huang, Zihao Huang, Yutao Zeng, Yunyao Mao, Banggu Wu, Qiyang Min, and Xun Zhou. Hyper-connections. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=9FqARW7dwB.

Zayd Muhammad Kawakibi Zuhri, Muhammad Farid Adilazuarda, Ayu Purwarianti, and Alham Fikri Aji. MLKV: Multi-layer key-value heads for memory efficient transformer decoding. *arXiv preprint arXiv:2406.09297*, 2024.

A WHITE-MATTER CONNECTIVITY

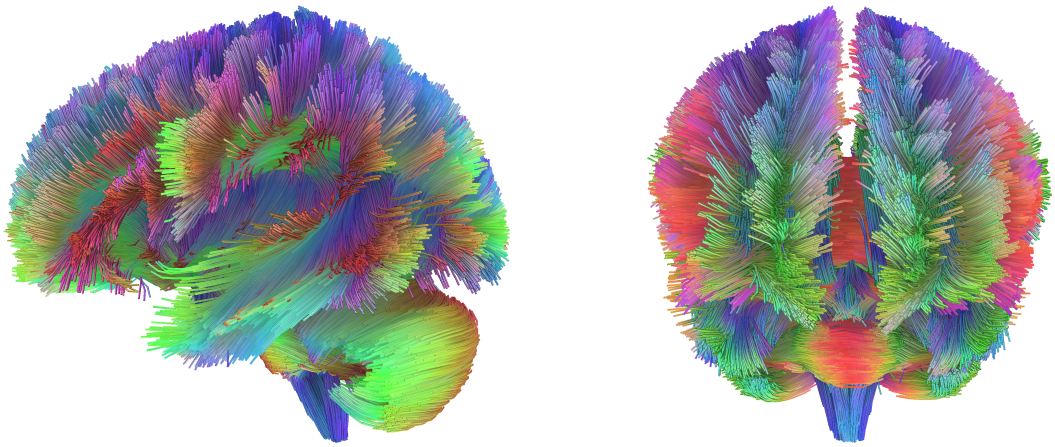

Figure 8: **Whole-brain white-matter tractography.** A population-averaged human structural connectome reconstructed from diffusion MRI, rendered as fiber tracts with DSI Studio (Yeh, 2025). The population-averaged template (Yeh et al., 2018) was built from Human Connectome Project data (Essen et al., 2013).

B KV POOL IMPLEMENTATION DETAILS

Router parameterization. The selected layers are spaced backward from $L-1$ at stride p and concatenated in increasing depth order. With $L' = \lceil L/p \rceil$, $\xi^K[i] \in \mathbb{R}^{L'D}$ and each channel's router has $W_j^{\alpha K} \in \mathbb{R}^{L \times L'D}$ and $b_j^{\alpha K} \in \mathbb{R}^L$. We stack these parameters into one linear layer with kL outputs, reshaped to $k \times L$.

Router initialization. Router weights start at zero. Biases initially select the top source for $k = 1$, sources satisfying $\ell \bmod k = j$ for $1 < k < L$, and source $\min(j+1, L-1)$ for $k = L$. In the main half-cache and full-cache models, selected bias entries are 0.25 and all others are zero.

Projection and cache indexing. Each channel has projection matrices $W_j^K, W_j^V \in \mathbb{R}^{H_{\text{kv}} d \times D}$, where H_{kv} is the number of KV heads and d is the head dimension. The boundary embedding is repeated across source layers before pool projection; its KV occupies slot zero, and real token i occupies slot $i+1$. The boundary has rotary position zero. Real-token positions start at one and reset for each packed document. The document mask always permits the boundary entry.

C EXPERIMENTAL DETAILS

We used `karpathy`'s `fineweb-edu-100b-shuffle` dataset and the Qwen3-0.6B-Base tokenizer (vocabulary 151,936). Table 3 gives additional architecture settings. Full-cache WhiteMatter uses two gradient-accumulation steps with global microbatch 64; all other main configurations use one accumulation step. The 1.3B parameter counts include the tied embedding/output matrix (272.3M parameters) once.

Table 3: **Additional pretraining settings.**

Setting	D=512	D=1792
Intermediate width	1536	5376
Query / KV heads	6 / 3	14 / 7
Head dimension	96	128
Optimizer steps	30,518	152,588

Optimization. Muon used momentum 0.95 and five Newton–Schulz steps; AdamW used $\beta_1 = 0.9$ and $\beta_2 = 0.95$. Both optimizers used a peak learning rate of 3×10^{-4}, 2% warmup, cosine decay to 10% of the peak, and weight decay of 0.1. Training used bfloat16 autocast with fp32 master weights on eight NVIDIA RTX A6000 GPUs. Before DDP all-reduce, each GPU clipped the gradient norm at 1.0.

Held-out perplexity at the larger scale. At approximately 1.3B parameters, vanilla and half-cache WhiteMatter achieved held-out perplexities of 13.51 and 12.93, respectively, under the evaluation protocol in §4.1.

D Inference benchmark details

We report median throughput over five runs after compilation and cache setup. Timings include the output head and greedy token selection; prefill computes only the final token's logits. Decoding uses CUDA graphs and excludes prefix preparation.

Peak memory is the maximum of two measurements during steady-state execution: the sampled device-memory peak, and PyTorch peak reserved memory plus a stable external-memory baseline.

E FLOP measurement details

We measured FLOPs with PyTorch at batch 1 and sequence length 2048, using causal attention and the schedules in §4.1. Decoding uses 2048 cached tokens. Counts include two operations per multiply-add and FusedKV's elementwise cache fusion. FusedKV prefill runs all prompt tokens through the source layers and only the final token through the reconstruction layers.

Table 4: **Per-token FLOPs for training, prefill, and decoding at D=512, in GFLOPs per token.**

Model	Decode	Prefill	Training
Vanilla 16L	0.179	0.142	0.444
FusedKV	0.203	0.071	0.429
LCKV w=4	0.173	0.935	1.421
LCKV w=7	0.175	0.738	1.174
WhiteMatter k=8	0.177	0.432	1.028
WhiteMatter k=16	0.184	0.467	1.111

F Convergence measurement details

F.1 Controlled prefill convergence

Reference model and schedules. The model in §4.3 has 4 layers, D=512, and k=4. It was trained with exact autoregressive execution for 800 steps at length 1024 and batch 96, totaling 78.6M tokens. We compared Jacobi iteration with cyclic groups and contiguous chunks, using $g \in \{2, 4, 8, 16, 32, 64\}$ for both partitions. Contiguous chunks are processed from left to right. Both grouped schedules evaluate tokens within each group in parallel using fixed KV and refresh the group's KV before processing the next group.

Pass selection. We used FP32 weights and arithmetic to evaluate on 192 unseen single-document sequences of length 2048. We pooled cross-entropy over all 393,024 next-token targets before exponentiating; the exact autoregressive reference perplexity was 165.44. For each schedule, we selected the first pass whose perplexity was at most 1.01 times this reference. Figure 9 shows the convergence trajectories.

Timing. Timing uses one A6000 GPU with compiled BF16 execution and batch size 32. We report median batch time divided by 32 after five warmups and 30 trials, or 10 full rollouts for autoregressive evaluation. Timing covers the decoder, excluding embeddings, final normalization, the output head,

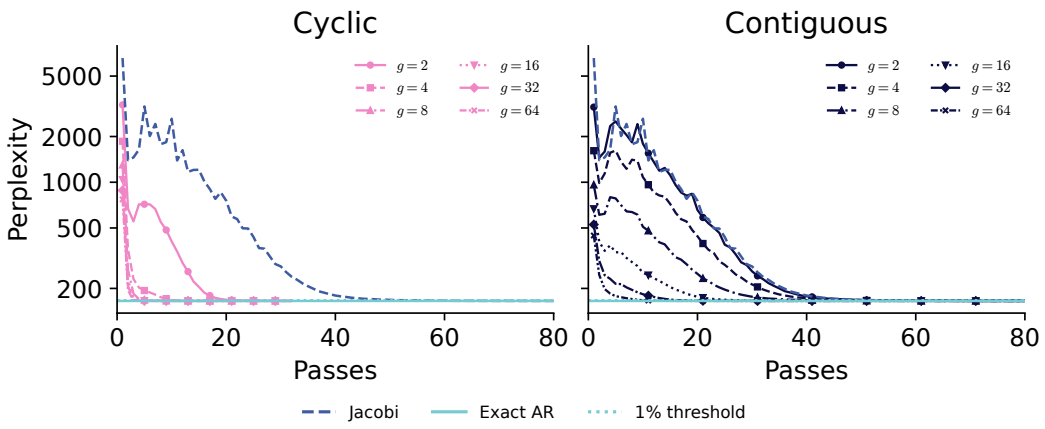

Figure 9: **FP32 prefill convergence on the autoregressively trained reference model.** Both partitions use $g \in \{2, 4, 8, 16, 32, 64\}$ groups. Curves show all evaluated passes. Horizontal lines mark exact autoregressive perplexity and the 1% threshold used for pass selection.

and token selection. Contiguous attention uses FlashAttention 2. Pass counts are selected in FP32; the BF16 timings measure execution cost at those counts.

G Convergence of a larger cyclic-trained model

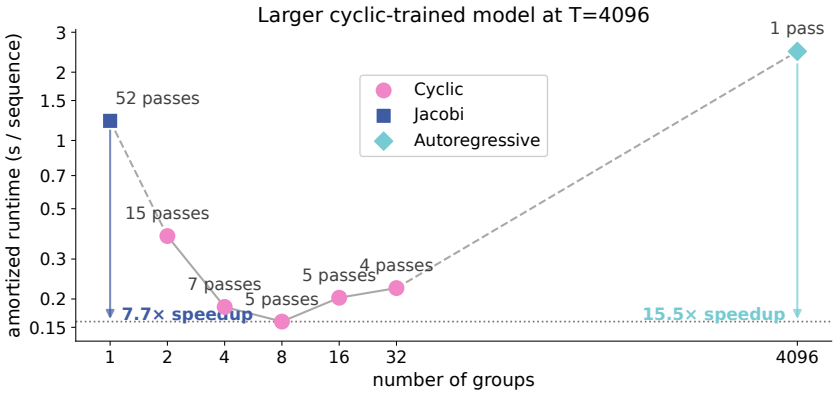

Figure 10: **Iteration-schedule runtime for a larger cyclic-trained model.** Cyclic iteration achieves the lowest runtime at the selected pass counts.

The 8-layer, D=1024, k=8 model was trained for 122,000 steps at global batch 8 and length 4096 (approximately 4.0B tokens) with cyclic g=8 iteration and frozen pretrained embeddings. Evaluation uses 192 held-out sequences at length 4096. We select the fewest passes reaching within 1% of the fp32 autoregressive perplexity (18.37) and measure runtime at batch 64.

At the selected pass counts, Jacobi takes 1.220 s/sequence for 52 passes and cyclic g=8 takes 0.159 s/sequence for five passes. The autoregressive rollout takes 2.470 s/sequence.

H Downstream evaluation details

Task scoring. In Tables 1 and 2, acc. n denotes length-normalized accuracy. LAMBADA perplexity is token-based; WikiText perplexity is word-based. BLiMP averages across 67 minimal-pair subtasks

(67,000 pairs), comparing grammatical and ungrammatical sentences. SciQ prepends the supplied support passage and scores four candidate answers. ReCoRD selects the highest-likelihood candidate completion and normalizes the predicted and accepted entities before comparison.

SQuAD generation. We use `squad_completion` (Arora et al., 2024) with the 2,984-example validation split of `hazyresearch/based-squad`. Answer matching is case-insensitive. Greedy generation permits up to 256 new tokens and stops at a blank line or EOS. The adapter retains up to 768 prompt tokens within the 1024-token context limit. FusedKV recomputes the prefix at each generated step.