SINGLE-LAYER, TWO AXES: COMMUNICATION AND REFINEMENT IN LOOPED TRANSFORMERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Looped Transformers increase computation by reusing parameters and outperform standard Transformers of the same size, but their architectures often change two things together: how a token is processed and which intermediate representations it can read. This coupling obscures the source of their gains. To disentangle the two, we introduce a Single-Layer framework that separates *refinement*, the choice of feed-forward operators, from *communication*, access to historical token–depth states. Unfolding recurrent depth into an addressable sequence of latent states lets one shared attention module support controlled changes to either axis. In 274M-parameter looped MoE language models with matched FFN compute, cross-depth access lowers held-out perplexity by 2.4–8.1% relative to same-depth access under all four refinement schedules, yet depth-specific expert banks still outperform a shared bank by 11.8–15.5%. Better communication and more effective parameter reuse are therefore distinct design questions, and they interact: the benefit of start injection depends on the available historical context. Varying the retained depths for recent and distant tokens shows that the preferred depth depends on token age and refinement, and that retaining all tested depths is worse than keeping the best one. These observations motivate a Single-Layer Transformer combining routed expert refinement with recent working memory and selective long-term memory. A compact 20M-parameter comparison at matched total capacity confirms the cross-depth advantage under both bank assignments, with a smaller gap between shared and depth-specific refinement. In the shared-bank model, $W = 8$ lowers perplexity by 7.9% relative to same-depth access and by 0.37% relative to full cross-depth access, while storing 84.8% fewer historical states.

1 INTRODUCTION

The Transformer organizes computation as a stack of physical layers (Vaswani et al., 2017). At each layer, a token is transformed by the current block and attends to representations at the corresponding depth. The layer index therefore determines both how far the token has been processed and which intermediate states its attention can directly access. Residual connections carry earlier information forward by folding it into a single running state, so attention sees only the current mixture. Computation depth and context depth consequently advance together.

Looped Transformers relax the connection between computation depth and parameter count. Repeatedly applying a shared block provides additional processing without introducing a new block at every step, as in Universal Transformers and recent recurrent language models (Dehghani et al., 2019; Geiping et al., 2025; Zhu et al., 2025). Such models outperform standard Transformers with the same number of parameters (Csordás et al., 2024; Bae et al., 2025; Zhu et al., 2025). Parameter reuse alone, however, leaves open what each step can read. Recurrent architectures also differ in attention access (Mohtashami et al., 2025), input injection (Geiping et al., 2025), feedback (Fan et al., 2020), and the assignment of feed-forward networks (Csordás et al., 2024; Bae et al., 2025). When these choices change together, an improvement could come from more useful processing, better access to earlier computation, or their interaction. This motivates our first question: *how do communication and refinement contribute to the performance of a looped model?*

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.

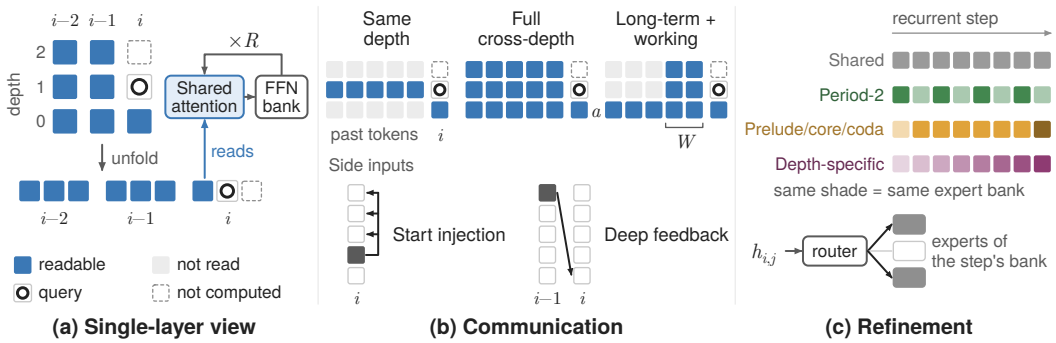


Figure 1: A single attention layer separates communication from refinement. (a) Unfolding the token–depth lattice into context positions makes intermediate states directly addressable by one reused attention module while keeping recurrent computation. Dashed states have not yet been computed and remain unreadable. **(b)** Communication selects historical states: matching depths, the full trajectory, or long-term anchors with recent working memory. The current-token prefix is held fixed; start injection and deep feedback provide additional directed inputs. **(c)** Refinement assigns FFN banks to successive updates; matching shades indicate reuse of the same bank. A router selects experts within the bank according to the state. Rows illustrate controls in the style of UT, Ouro, Huginn and a layer stack, all built on the same attention module.

We study this question through a *Single-Layer* framework. Recurrent states need not occupy distinct physical layers to remain accessible: unfolding each token's trajectory makes token position and recurrent depth coordinates of one attention context. A shared attention module reads selected states, defining *communication*; the assignment of FFN operators defines our *refinement* control. This common interface retains recurrent computation while allowing either choice to change independently (Figure 1). An earlier token can expose states from several recurrent depths because its whole trajectory has already been computed (Figure 1a). Cross-depth attention thus expands what a token can read without adding refinement steps or violating causality.

In looped MoE models with about 274M parameters, we partition a fixed expert pool into shared, periodic, prelude/core/coda, or depth-specific banks, and cross them with three historical-access rules at fixed expert width and per-step activation count. Full cross-depth access lowers perplexity by 2.4–8.1% over same-depth access under all four schedules, yet depth-specific banks outperform the shared bank by 11.8% under same-depth and 15.5% under full cross-depth access. Start injection's 2.5% gain for the prelude/core/coda control under same-depth access vanishes under full access. These interactions show why processing and information access must be examined together.

Access to more historical computation raises our second question: *which representations need to remain readable?* Exposing a completed token's entire trajectory increases attention work and storage, and states useful nearby may lose their value far into the future. We therefore vary retained depths separately in recent working memory and distant history. The preferred depth depends on token age and refinement, and keeping all tested depths raises perplexity by at least 10.5% over the best one.

The resulting Single-Layer Transformer combines routed expert refinement with rich recent trajectories and selected long-term representations. A compact 20M-parameter comparison confirms cross-depth gains under both bank assignments, while the shared-bank deficit narrows. With a shared bank and an 8-token working window, selective memory stores 84.8% fewer historical states and lowers perplexity by 7.9% and 0.37% relative to same-depth and full cross-depth access, respectively. Training uses parallel updates, and evaluation uses exact token-by-token decoding.

Our contributions are:

- **Single-Layer framework.** Unfolding recurrent depth into one attention context lets communication and refinement change independently at fixed parameters and FFN compute.
- **Two controlled studies.** Cross-depth access helps every refinement schedule, yet depth-specific banks remain stronger. The best retained depth depends on token age and refinement.
- **Single-Layer Transformer.** Routed refinement with working and long-term memory lowers perplexity by 7.9% over same-depth access and stores 84.8% fewer historical states.

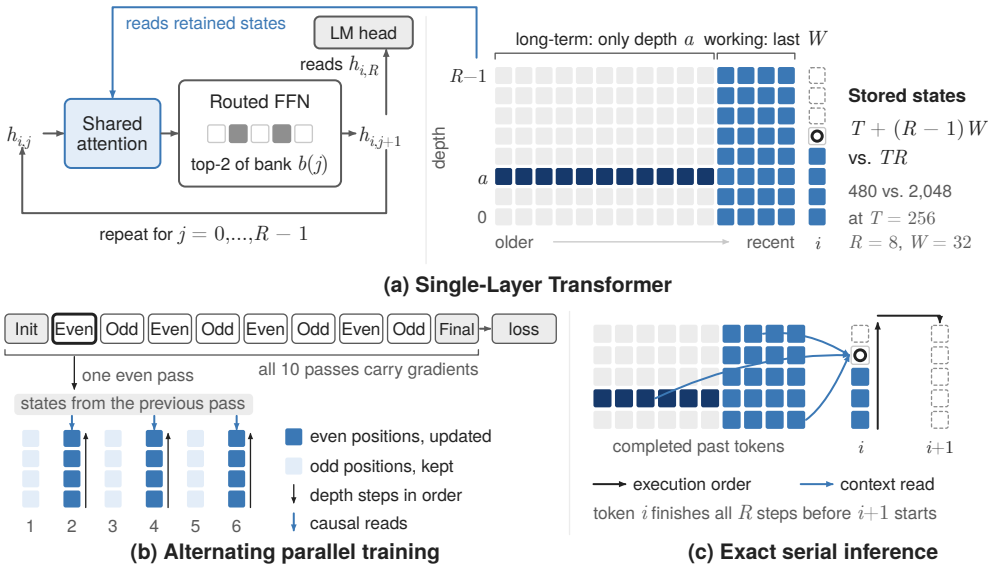


Figure 2: Single-Layer architecture, alternating training, and serial inference. (a) The compact model reuses one attention–MoE block for eight steps, activating two experts per step. Recent tokens keep complete trajectories and older tokens keep one anchor depth; grid sizes are illustrative. (b) Full initialization, four even/odd rounds and full readout give ten gradient-tracked passes. In the even pass shown, even positions update in parallel with sequential depth while odd positions keep their states; the next pass reads the refreshed states. (c) Inference completes token i before $i + 1$, reading the retained states of earlier tokens. Black arrows show execution, blue arrows context reads, and the ring the current query. Appendix A gives the compact-model configuration, and Appendix F specifies the schedules of the other studies.

2 Related Work

Prior work connects parameter reuse, intermediate-state access and historical memory. We vary these choices separately within a common model to study their interaction.

Recurrent computation and expert reuse. Universal Transformers share a transition across depth; Huginn and Ouro organize recurrence through different parameter schedules (Dehghani et al., 2019; Geiping et al., 2025; Zhu et al., 2025). Recent looped-MoE studies examine sparse processing and the allocation of attention and feed-forward computation (Lee et al., 2026; Lin et al., 2026; Wang et al., 2026a). We compare expert-bank assignments while holding attention and its readable history fixed, separating specialization from information access.

Communication across computational depth. Input injection, loop-time memory and token–iteration attention provide different routes to earlier computation (Geiping et al., 2025; Knupp et al., 2026; Wang et al., 2026c; Fu et al., 2025). Latent reasoning and feedback architectures also pass intermediate states to later computation (Hao et al., 2024; Wang et al., 2026b). These mechanisms motivate our communication axis. Our controls borrow them to ask when additional access helps and whether its benefit changes with the refinement schedule.

Selective recurrent memory. Parallel Loop Transformer combines shared global and loop-specific local context (Wu et al., 2025). MELT updates a shared memory, while Looped Latent Attention compresses loop-indexed representations (Conchello Vendrell et al., 2026; O'Neill & Reid, 2026). We select readable depths according to token age and compare this selection across refinement schedules. This addresses which states persist, complementing compression of each retained state.

3 Preliminaries

A single physical block can produce many representations of a token. We index these states by token position i and recurrent depth j, writing $h_{i,0} = \mathrm{Emb}(x_i)$ and $h_{i,j+1}$ for the output of update j.

Physical depth counts separately parameterized blocks; recurrent depth counts applications. Keeping these notions distinct lets us study information access without changing the number of updates.

3.1 Communication and refinement

An update can receive information from two sources: completed states of earlier tokens and the current token's already computed trajectory. Define

$$\mathcal{X}_{i,j} \subseteq \{(m,n) : m < i,\ n \in \mathcal{D}\}, \qquad \mathcal{T}_{i,j} = \{(i,n) : 0 \le n \le j\}, \tag{1}$$

where $\mathcal{D}$ is the set of historical depths written by the policy. The complete context is $\mathcal{C}_{i,j} = \mathcal{X}_{i,j} \cup \mathcal{T}_{i,j}$. With shared attention parameters θ_A and feed-forward assignment ϕ_j, the update is

$$h_{i,j+1} = B_{\theta_A,\phi_j}(h_{i,j}; \{\mathrm{KV}(h_{m,n}) : (m,n) \in \mathcal{C}_{i,j}\})\,. \tag{2}$$

Communication varies the historical set $\mathcal{X}_{i,j}$; *refinement* varies the feed-forward assignment ϕ_j. Attention also transforms the residual state; refinement here names only the controllable FFN assignment. Both axes retain the same current-token set $\mathcal{T}_{i,j}$.

3.2 Exact causal execution

Exact execution finishes all R updates of token i before starting token $i+1$. A current query can therefore read a deeper representation of an earlier token without seeing a future token. In the first study, $\mathcal{D} = \{0, \dots, R-1\}$, and the two reference policies are

$$\mathcal{X}^{\mathrm{same}}_{i,j} = \{(m,j) : m < i\}, \tag{3}$$

$$\mathcal{X}^{\mathrm{full}}_{i,j} = \{(m,n) : m < i,\ n \in \mathcal{D}\}. \tag{4}$$

The terminal state $h_{i,R}$ predicts the next token. A policy that also retains this state must explicitly project and write it for subsequent tokens. Terminal anchors are therefore specified separately from the default historical set $\{0, \dots, R-1\}$ (Section 4.2).

We also study two directed side inputs. Start injection supplies the current token's prelude output to later core updates. Deep feedback supplies the preceding token's terminal state to the current token's first update. These add state inputs without changing the attention set; their write operator is given in Appendix A. Independent sequences can be batched during exact evaluation, but the token–depth execution order within each sequence remains unchanged.

4 Method

Single-Layer provides a common attention interface for two decisions: which historical states to receive and which experts to apply. We first specify the block and its refinement controls, then introduce age-dependent historical access, and finally describe training on the resulting cross-token graph. Figure 2 connects these choices to their execution.

4.1 Shared attention and routed refinement

The block acquires information through shared attention and processes the result through an expert bank. Given the attention-updated state, a router selects operators from bank $b(j)$ and weights their outputs by routing confidence. The architecture study uses one routed and one always-on expert; the compact model selects two routed experts with no always-on expert. The bank schedule constrains the available operators; the token state determines the choice within that bank. A single shared bank therefore still performs state-dependent operator selection at every step.

This distinction gives four refinement controls with one resident expert pool. The shared schedule uses the same bank at every depth, the periodic schedule alternates banks, the prelude/core/coda schedule separates entry, repeated processing and exit, and the depth-specific schedule assigns a bank to each update. Resident capacity and per-step expert width are fixed within each study; the number of distinct parameters visited across a trajectory can differ.

These schedules are inspired by UT, Ouro, Huginn and a depth-specific stack, respectively (Dehghani et al., 2019; Zhu et al., 2025; Geiping et al., 2025). All four share one attention module, including the depth-specific control. Appendix A details bank counts, residuals, normalization and routing.

Token position and recurrent depth describe different coordinates. We apply rotary positions separately in token and depth subspaces, using the same coordinates during training and inference. Only the terminal state is read by the tied language-model head. The loss combines next-token cross-entropy with expert load balancing; intermediate states receive no separate prediction targets.

4.2 Long-term and working-memory communication

Full cross-depth access makes historical computation available; how long each representation stays useful is a separate choice. We make this choice explicit by assigning different readable depth sets to recent and distant tokens. Let W denote the number of preceding tokens in working memory, and let $\mathcal{A}_{\mathrm{W}}(j)$ and $\mathcal{A}_{\mathrm{L}}(j)$ denote the depths readable by query depth j inside and outside this window, respectively. The selective historical set of query (i, j) is

$$\begin{aligned}\mathcal{X}^{\mathrm{sel}}_{i,j} = &\{(m,n) : 0 \le m < i - W,\ n \in \mathcal{A}_{\mathrm{L}}(j)\} \\ &\cup \{(m,n) : \max(0, i-W) \le m < i,\ n \in \mathcal{A}_{\mathrm{W}}(j)\}.\end{aligned} \tag{5}$$

The current token still receives $\mathcal{T}_{i,j}$. All sets are deduplicated, so a coordinate selected by two different rules enters the attention context exactly once.

An anchor $z_{i,a} = h_{i,a}$ is the retained state of token i at depth a. The formulation includes complete recent trajectories with one old-token anchor, as well as a few boundary states at either timescale. When the terminal state is retained, R belongs to the eligible depth set. The compact selective policy takes $\mathcal{A}_{\mathrm{W}}(j) = \{0, \dots, R-1\}$ and $\mathcal{A}_{\mathrm{L}}(j) = \{a\}$: recent tokens expose complete trajectories, whereas distant tokens retain one selected depth. The selection study compares added boundary states in recent and distant history, and the compact-model study varies W at a fixed anchor. For the single-anchor policy, changing a at fixed W preserves the number of stored states. In the boundary-addition study, overlap with the segment-entry path is deduplicated and counted explicitly.

If d_{L} and d_{W} count the distinct depths retained across queries for old and recent tokens, the historical state count after T tokens, excluding the current token, is

$$N(T) = (T - W)_+ d_{\mathrm{L}} + \min(T, W) d_{\mathrm{W}}. \tag{6}$$

This counts persistent states, and old-token anchors still grow with context length. Appendix D accounts for attention reads and training activations separately.

4.3 Parallel training and serial inference

Reading completed past-token trajectories introduces a serial dependency: a shallow update may need a deeper state of the previous token. Exact token-major execution follows this dependency directly. Training instead refreshes historical state estimates through parallel updates, while retaining exact execution as the evaluation target (Figure 2b,c).

A full initialization pass constructs a bank $Z^{(0)}$. Within each subsequent pass, the selected tokens update in parallel, each executing its recurrent depth chain in order. Causally masked historical reads use a source bank fixed during that pass; the current token reads its newly computed prefix. An update $\mathcal{J}_{\theta,U}$ replaces the states of token subset U and preserves all others. In each round k, the compact study first updates even and then odd token positions:

$$Z^{(k+\frac{1}{2})} = \mathcal{J}_{\theta,\mathrm{even}}(Z^{(k)}; x), \qquad Z^{(k+1)} = \mathcal{J}_{\theta,\mathrm{odd}}(Z^{(k+\frac{1}{2})}; x). \tag{7}$$

Thus the odd pass reads the bank refreshed by the even pass. After four rounds, a final full pass produces the next-token predictions. The full initialization, eight parity passes and full readout give ten forward passes, and backpropagation runs through all of them, including the fixed source values. Each updated token undergoes all R recurrent steps. The axis study uses a differentiated suffix of a longer alternating schedule; the boundary study uses two eight-step blocks with two Jacobi sweeps. Appendix B gives the gradient scope and checks; Figure 2b illustrates the compact training schedule.

At inference, completed states of preceding tokens replace the estimates. Every checkpoint is evaluated on this exact graph, whose quality ranks topologies; paired parallel scores measure the training approximation error, and the update-count study checks it in its tested configuration.

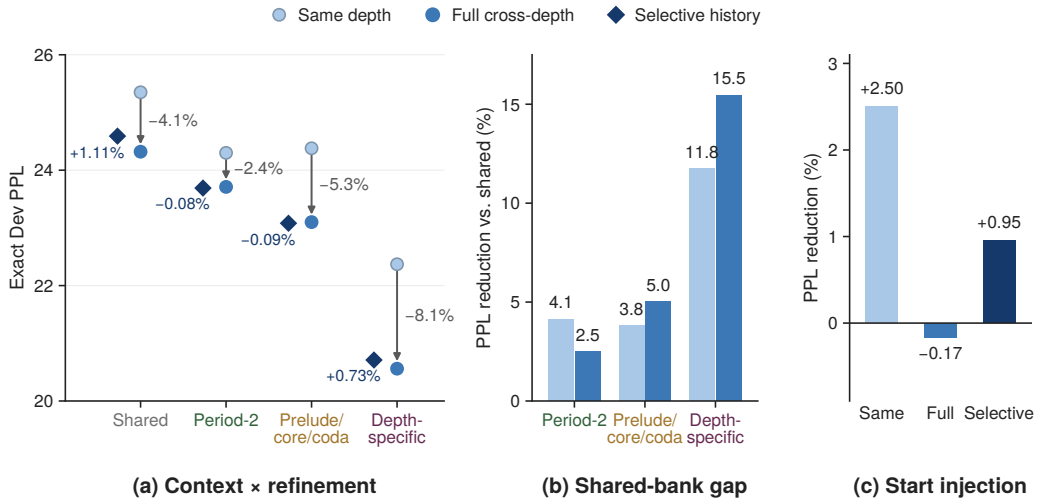


Figure 3: Historical access and expert reuse have distinct, interacting effects. (a) Exact Dev PPL for each refinement schedule without side inputs; arrows give the change from same-depth to full cross-depth access, and diamonds mark selective history beside the full-context endpoint, labeled with its change relative to that endpoint. (b) PPL reduction relative to the shared bank under the same context; the depth-specific advantage persists under full cross-depth access. (c) Start-injection gain relative to injection-off for prelude/core/coda refinement. All panels derive from the same single-run axis-study endpoints (approximately 274M resident parameters, $R = 16$), so the contrasts share their training runs.

5 Experiments

We first separate communication from refinement, then ask which historical states are useful enough to retain. A compact-model study tests the resulting design under a smaller resident parameter budget.

Experimental settings. The first two studies use approximately 274M resident parameters and 20M active parameters per recurrent step, with $R = 16$ and context 256. The axis study trains for 450M tokens; the memory study trains for 400M. The final study uses approximately 20M *total* parameters, $R = 8$, and 400M tokens. Its ten gradient-tracked passes comprise a full initialization, four even/odd rounds and a final full pass. All use hidden width 512, eight query heads, four K/V heads, a 16K vocabulary, and the same FineWeb-Edu-derived corpus. Comparisons are matched within each study, and absolute PPL values are compared only within a study. Appendix F specifies the larger-model training contracts, and Appendix A gives the compact-model configuration.

Evaluation. Exact Dev perplexity is the exponential of mean next-token cross-entropy over fixed held-out windows. Parallel Dev PPL uses those same windows and checkpoint, changing only the execution graph. The available endpoints are single-run results. Paired execution scores and training configurations are reported in Appendices B and F.

5.1 Separating communication and refinement

Cross-depth access helps every tested refinement schedule. Full cross-depth access reduces exact Dev perplexity by 2.4–8.1% relative to same-depth access across all four plain schedules. Figure 3a plots each schedule under the three contexts: each arrow changes communication from same-depth to full cross-depth access at a fixed schedule, and the horizontal axis varies refinement. The paired same-depth and full cross-depth models have the same resident expert pool and perform the same number of recurrent FFN applications. The gain therefore comes from historical access. Extra attention work is accounted for separately in Appendix D.

Bank assignment still matters under full access. Depth-specific refinement remains the strongest endpoint under both base contexts, outperforming the shared bank by 11.8% under same-depth and 15.5% under full cross-depth access (Figure 3b). Comparing columns within a row of Table 1 tests specialization under identical historical access. Richer context even widens the depth-specific

Table 1: Communication and refinement on a common Single-Layer framework. Exact Dev PPL at approximately 274M resident parameters, $R = 16$, and 450M training tokens. Columns give bank count $\times$ routed experts per bank; one additional always-on expert is common to the configurations. Dashes mark unavailable conditions; shaded rows without side inputs are plotted in Figure 3a.

Context	Side input	Shared 1×48	Period-2 2×24	Prelude, core and coda 3×16	Depth-specific 16×3
Same depth	None	25.35	24.30	24.38	22.37
	Start injection	—	—	23.77	—
	Deep feedback	25.39	24.31	24.38	22.34
Full cross-depth	None	24.32	23.71	23.10	20.56
	Start injection	—	—	23.14	—
	Deep feedback	24.34	23.75	23.08	20.59
Selective history	None	24.59	23.69	23.08	20.71
	Start injection	—	—	22.86	—
	Deep feedback	24.61	23.71	23.13	20.72

advantage in this setting. This motivates the compact study: can a smaller shared bank use its broad operator choice more effectively at a fixed training budget?

The benefit of start injection depends on historical access. Start injection improves same-depth prelude/core/coda refinement by 2.5%, whereas its full cross-depth counterpart moves from 23.10 to 23.14 PPL. Figure 3c also includes selective history, where injection improves PPL by 0.95%; each contrast fixes refinement and context. The value of the auxiliary input therefore depends on the available context. Injection supplies the current token's early state, whereas historical attention supplies previous tokens. Table 1 also reports the deep-feedback variants.

Selective history nearly matches full access. Selective-history endpoints stay within 1.1% of full cross-depth PPL for all four schedules, with small improvements for the periodic and prelude/core/coda schedules. Figure 3a places each selective marker beside its schedule's full-context endpoint, separating the cost of restricting history from the larger differences between schedules. Restricted history therefore reaches near-full quality in this setting.

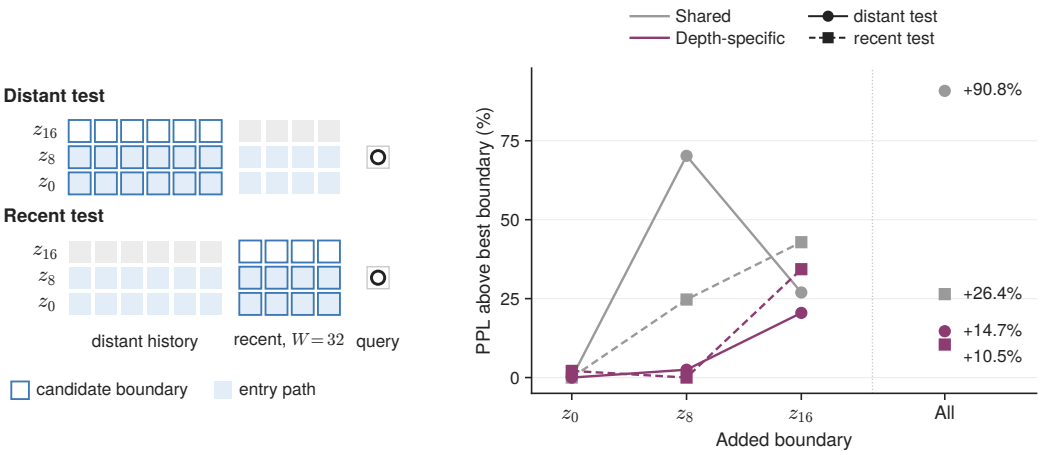
This motivates testing *which* states can be removed. We next vary the retained boundaries and their temporal range; the compact-model study then measures the resulting quality–cost tradeoff.

5.2 Selecting long-term and working memory

The entry state is the best tested long-term anchor. Adding the entry state z_0 to distant history gives the best exact PPL among the tested additions for both refinement controls. Figure 4a isolates the intervention: add a boundary inside or outside $W = 32$, while keeping the other region's query-matched entry path. Panel b compares z_0, z_8, z_{16}; overlapping coordinates are deduplicated.

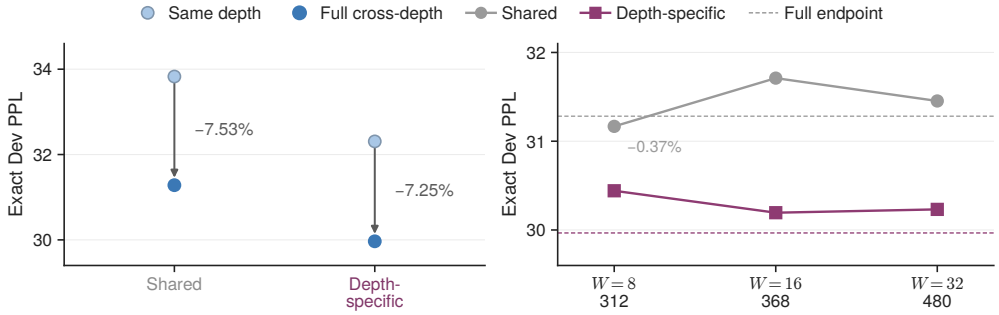
The preferred depth depends on both token age and refinement. Depth-specific refinement favors z_0 outside the window but z_8 inside it; the shared bank favors z_0 in both regions. The deepest state, z_{16}, is worse than the preferred boundary in every slice: the best state to retain is shallower.

Retaining all boundaries is worse than the best single one. Retaining all three boundaries raises PPL by 10.5–90.8% relative to the best single boundary (Figure 4b). In this training setting, broader access can hurt, which motivates separate selection for working memory and distant history. These anchor-to-anchor comparisons share a two-sweep block-Jacobi schedule; the differently trained no-extra-anchor reference remains separate (Table 6). The compact study next tests whether a shorter working window preserves quality with fewer stored states.



(a) Matched interventions **(b) Cost of each retained boundary**

Figure 4: The useful retained depth depends on token age and refinement. (a) Each intervention varies added boundaries in one history region, with the other region's entry path fixed. (b) Exact Dev PPL of each added boundary relative to the best single boundary of the same slice, $100(\mathrm{PPL}/\mathrm{PPL}_{\mathrm{best}} - 1)$; each line reaches zero at its preferred depth. All denotes the three boundaries $\{z_0, z_8, z_{16}\}$. These single-run, $R = 16$ comparisons share one training schedule. Table 6 gives complete values and separate exact-trained references.



(a) Context × bank **(b) Working window**

Figure 5: Compact-model communication gains and the working-window tradeoff. (a) Exact Dev PPL under same-depth and full cross-depth access for each bank; arrows give the relative change. (b) Selective-history PPL versus working window at fixed $a = 2$; tick labels also give the retained historical states. Dashed lines mark each bank's full-context endpoint, and shared $W = 8$ is 0.37% below it. All ten points are single-run compact endpoints from Table 2; the best window differs by bank.

5.3 Compact-model validation

Cross-depth gains persist at matched total capacity. At approximately 20M total parameters, full cross-depth access reduces exact Dev PPL by 7.53% for the shared bank and 7.25% for depth-specific banks relative to their same-depth controls (Figure 5a). Table 2 crosses two bank assignments—shared (UT-style MoE) and depth-specific—with same-depth and full cross-depth access. The four controls have the same resident expert pool, Top-2 routing, recurrent depth, data budget and ten-pass alternating schedule. Comparing bank assignments within each context tests refinement; comparing contexts within each bank assignment tests communication. This small factorial comparison revisits the shared bank's deficit from the first study at a smaller scale.

The shared-bank deficit narrows but persists. Depth-specific banks improve on the shared bank by 4.49% under same-depth and 4.21% under full cross-depth access, versus 11.8% and 15.5% in the larger-capacity study. The ranking holds, while its magnitude depends on the model and training setting. The smaller study also changes expert width, recurrent depth and routing along with capacity. Paired execution checks preserve both the communication and bank rankings for the compact checkpoints, with graph gaps of at most 0.72% in PPL (Appendix B).

Table 2: Compact-model validation of bank reuse and selective memory. All configurations have approximately 20M total parameters, $R = 8$, context 256 and 400M training tokens. Training uses one full initialization pass, four even/odd rounds and one final full pass, all carrying gradients. Bank columns report exact Dev PPL; the first two rows form four core controls. Window variants fix the distant anchor and bank assignment. Inference GF/token analytically counts the matrix multiplications in the shared-bank strict token-major decoder, averaged over 256 positions; the same-depth kernel computes dense attention before masking. Both GF/token and the historical-state counts are analytic, and the state counts exclude the current token.

Communication	**Shared** 1×24	**Depth-specific** 8×3	**Inference** GFLOPs per token	**Historical states**
Same depth	33.83	32.31	0.061	2,048
Full cross-depth	31.28	29.97	0.061	2,048
Long-term + working				
$W = 8, a = 2$	31.17	30.44	0.047	312
$W = 16, a = 2$	31.71	30.19	0.048	368
$W = 32, a = 2$	31.45	30.23	0.050	480

Selective memory preserves near-full-context quality. For the shared bank, $W = 8, a = 2$ lowers exact Dev PPL by 7.87% relative to same-depth access and by 0.37% relative to full cross-depth access (Figure 5b). The policy reduces the analytic count of historical records from 2,048 to 312, an 84.8% reduction at the evaluated context length. With the anchor fixed, $W = 16$ and $W = 32$ achieve exact Dev PPL of 31.71 and 31.45 for the shared bank, and 30.19 and 30.23 for depth-specific banks, retaining 368 and 480 historical states, respectively (Table 2). Across these single-seed runs, the best window is $W = 8$ for the shared bank and $W = 16$ for depth-specific banks.

6 Conclusion

We introduced Single-Layer as a common framework for studying communication and refinement in looped Transformers. Representing recurrent computation as an addressable token–depth trajectory makes it possible to change the history supplied to an update without changing its feed-forward schedule, or to change expert reuse under fixed historical access. Cross-depth access improves every tested refinement schedule, while the benefit of start injection changes with the available context. Depth-specific expert banks nevertheless remain stronger than a shared bank in both model settings. This distinction ties differences between recurrent architectures to specific choices about information access and processing, beyond the number of repeated blocks.

Our second study asks how much of the exposed history is needed. The best retained boundary depends on token age and refinement, and under the tested training schedule a selected subset beats exposing all tested boundaries. The resulting model combines a routed expert bank with rich working memory and selected long-term representations. The compact comparison confirms the cross-depth advantage under both bank assignments, while the shared-bank deficit narrows. Its $W = 8$ selective-memory model stores 84.8% fewer historical records, with exact Dev PPL 0.37% below full cross-depth access. This supports a quality–storage tradeoff at the tested scale. Exact token-major evaluation and paired graph checks distinguish these architectural results from the parallel approximation used in training. The design principle is to select processing operators and historical access together, while making each independently controllable.

AI Use Statement

We used ChatGPT and Codex (OpenAI) to assist with literature search and synthesis, discussion of the conceptual framework and mathematical formulations, hypothesis refinement, experimental design, implementation and debugging of training and evaluation code, data-preparation scripts, and interpretation of experimental results. These tools also assisted with manuscript organization, drafting, translation and editing, and with the design and production of scientific figures. Generative image tools were used to explore schematic figure designs before their editable reconstruction. The authors directed these activities and iteratively revised the resulting text, code and figures. AI-generated

suggestions are not empirical evidence or independent verification of a claim. The authors are responsible for verifying the results, citations and AI-assisted artifacts, and for the final submission.

Ethics Statement

This work studies language-model architectures using an existing FineWeb-Edu-derived corpus and held-out evaluation data; the experiments do not involve recruiting human participants. Models trained on web-derived text may reproduce harmful content, social biases or personal information present in their training data. Changes to communication and refinement do not by themselves address these risks, and the reported quality and efficiency measurements should not be interpreted as evidence of safety for deployment. Use and redistribution of the data and trained models must respect the applicable access conditions and licenses.

Reproducibility Statement

Sections 3 and 4 define the communication graphs, refinement schedules, and training and inference procedures. Appendix A describes the controlled model configurations, auxiliary paths and parameter accounting. Appendix B specifies the training schedules, gradient scopes, and paired parallel/exact evaluation protocol. Appendix C specifies the interpretations and additional controls attached to the architectural comparisons. Appendix D provides state accounting, and Appendix E states the assumptions behind the graph-discrepancy bound. Appendix F documents the larger-capacity axis and memory studies and distinguishes their training contracts from the compact validation and the alternating-update pilot. These details distinguish architectural choices from execution approximations when reproducing the comparisons.

References

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise LoRA. In *International Conference on Learning Representations*, 2025.

Victor Conchello Vendrell, Arnau Padres Masdemont, Niccolò Grillo, et al. Memory-efficient looped transformer: Decoupling compute from memory in looped language models. *arXiv preprint arXiv:2605.07721*, 2026.

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

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

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

Tianyu Fu, Yichen You, Zekai Chen, Guohao Dai, Huazhong Yang, and Yu Wang. Think-at-hard: Dynamic looped transformers for improved reasoning. *arXiv preprint arXiv:2511.08577*, 2025.

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

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. *arXiv preprint arXiv:2412.06769*, 2024.

Jonas Knupp, Jan Hendrik Metzen, Jeremias Bohn, Georg Groh, and Kristian Kersting. Depth-recurrent attention mixtures: Giving latent reasoning the attention it deserves. *arXiv preprint arXiv:2601.21582*, 2026.

Ryan Lee, Jacob Biloki, Edward J. Hu, and Jonathan May. Sparse layers are critical to scaling looped language models. *arXiv preprint arXiv:2605.09165*, 2026.

Ruhai Lin, Yiyang Guo, Rui-Jie Zhu, Hao Ye, and Jason K. Eshraghian. Allocating recurrent compute in looped language models. *arXiv preprint arXiv:2608.18230*, 2026.

Amirkeivan Mohtashami, Matteo Pagliardini, and Martin Jaggi. CoTFormer: A chain-of-thought driven architecture with budget-adaptive computation cost at inference. In *International Conference on Learning Representations*, 2025.

James O'Neill and Fergal Reid. Looped latent attention: Cross-loop KV compression for looped transformers. *arXiv preprint arXiv:2607.15456*, 2026.

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*, 2017.

Shaowen Wang, Ge Zhang, Kairong Luo, et al. SMELT: Scaling laws for compute-matched MoE looped transformers. *arXiv preprint arXiv:2609.01343*, 2026a.

Xi Wang, Ziyang Cai, Zheng Zhan, Harry Dong, Ying Fan, Gustavo de Rosa, Tim Pearce, and John Langford. Full-bandwidth transformer. *arXiv preprint arXiv:2608.08888*, 2026b.

Yuxiang Wang, Kunyu Feng, Yingda Shen, Haoning Xu, Junyu Wang, and Zhizheng Wu. Recurtrace: Adaptive latent reasoning with loop-time memory. *arXiv preprint arXiv:2609.03379*, 2026c.

Bohong Wu, Mengzhao Chen, Xiang Luo, et al. Parallel loop transformer for efficient test-time computation scaling. *arXiv preprint arXiv:2510.24824*, 2025.

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

A Implementation and Evaluation Details

A.1 Compact-model configuration

The compact study matches total resident capacity across the different expert pools. The plain configurations have 20,203,648 resident parameters, including the token embedding and all stored experts. The attention module is shared at every recurrent step, even in the depth-specific control.

Schedule	Bank rule $b(j)$	Expert allocation
Shared	0	1×24
Depth-specific	j	8×3

Every step selects two experts from its eligible bank, using router confidence to scale their outputs. The distinct parameters visited over a whole token trajectory can differ between schedules, because Top-2 per step can reach different experts at different steps. We distinguish resident parameters, active parameters per step, and the routed union over the trajectory. The compact comparison matches the resident count and the number and width of selected experts per step; comparing trajectory-wide active unions would need routing traces beyond these endpoints.

All variants use Q/K normalization, attention head gating, post-block normalization, and separable token–depth rotary positions. Depth modulation and router depth bias are off. Expert load balancing is computed per depth over the batch, with coefficient 0.01. The tied language-model head reads the final recurrent state. Intermediate states receive no separate next-token targets.

Table 3: Compact study configuration. Shared settings for the four same/full-context and shared/depth-specific controls. The selective-memory extension keeps these settings fixed.

Setting	Value	Setting	Value
Resident parameters	20,203,648	Recurrent depth	8
Hidden width	512	Query / K/V heads	8 / 4
Head dimension	64	Vocabulary	16,000
Resident expert slots	24	Expert width	304
Selected experts/step	2	Always-on shared experts	None
Context length	256	Training tokens	400,031,744
Global batch	128 sequences	Initial seed	42
Even/odd rounds	4	Forward / gradient passes	10 / 10
Full / parity passes	2 / 8	Gradient detachment	None

Optimization and data. AdamW uses peak learning rate 6×10^{-4}, 10M warmup tokens, minimum learning-rate ratio 0.1, $(\beta_1, \beta_2) = (0.9, 0.95)$, weight decay 0.1, and gradient clipping at 1.0. The global batch and token order are fixed; microbatch size and accumulation may vary with hardware without changing the global optimization batch. The initial deployment uses microbatch 64. Checkpoints and validation are scheduled every 50M tokens. Exact Dev evaluation uses 128 fixed context-256 windows, initially batched in groups of 16 independent sequences. The ten completed compact endpoints use seed 42 and checkpoint step 12,208, after 400,031,744 training tokens. Each Dev evaluation covers 32,768 tokens. The corpus is FineWeb-Edu-derived with a 16K tokenizer vocabulary; the available result records omit the corpus revision and tokenizer checksum. Each configuration has one seed, and independent training-sample PPL is unavailable. The Dev comparisons serve architectural analysis; a separate final-test evaluation remains future work.

Memory-selection configuration. The shared-bank selective control was configured with $W = 32$ and $a = 2$, retaining all eligible depths for recent tokens and one anchor for every distant token; $W = 16$ and $W = 8$ were added later at the same anchor. It follows the same 20M, $R = 8$, ten-pass alternating contract. The compact memory extension varies the working window without repeating the full boundary-selection matrix. The completed shared-bank and depth-specific $W = 32$, $W = 16$ and $W = 8$ endpoints use the same token budget and seed as the four core controls. Run identifiers and full-precision scores are in the source-data file `results/compact_results.csv`. The $R = 16$ boundary-state results in Table 6 use the separate contract in Appendix F.

A.2 Coordinates and auxiliary state inputs

Token and recurrent-depth coordinates occupy separate head subspaces:

$$\mathrm{RoPE}_{2\mathrm{D}}(u; i, j) = \left[\mathrm{RoPE}(u^{\mathrm{tok}}; i); \mathrm{RoPE}(u^{\mathrm{dep}}; j)\right]. \tag{8}$$

Start injection and deep feedback use the write family

$$D(z, s) = \rho \odot z + \eta \odot Ws, \qquad \eta = \mathrm{softplus}(b), \quad \rho = \exp[-\eta \odot \exp(\ell)]. \tag{9}$$

Start injection takes $h_{i,1}$ as s before subsequent core updates. Deep feedback takes $h_{i-1,R}$ before the first update of token i, with a fixed initial state at the sequence boundary. Each auxiliary module introduces a 512×512 projection and two 512-dimensional vectors, or 263,168 parameters. This is about 0.096% of the larger resident pool, so the comparison is parameter-matched to within 0.1%. The compact four-control study runs without side inputs.

B Training and Exact Execution

B.1 Full-gradient alternating training for the compact model

The compact study initializes all token states, performs four rounds of even/odd updates, and finishes with an all-token prediction pass. Each round contains two passes: even positions are updated first, then odd positions read the resulting bank. Positions within the selected parity run in parallel; the $R = 8$ depth updates within each token remain sequential. The initialization runs inside every training step. All ten passes and their source-bank dependencies remain in the gradient graph, so the loss gradient reaches every pass back to the state initialization.

Algorithm 1: Ten-pass, full-gradient alternating training for Table 2.

Input: Tokens x, $R = 8$, even/odd index sets E, O, communication rule $\mathcal{C}$, refinement schedule $b(j)$.

$Z \leftarrow \mathrm{Seed}_\theta(x)$; // full initialization, pass 1

for $k \leftarrow 1$ **to** 4 **do**

  $(H_E, Z) \leftarrow \mathrm{Scan}_\theta(Z, E)$; // even-token pass

  $(H_O, Z) \leftarrow \mathrm{Scan}_\theta(Z, O)$; // odd-token pass

$(H, Z) \leftarrow \mathrm{Scan}_\theta(Z, E \cup O)$; // full readout, pass 10

$\mathcal{L}_{\mathrm{CE}} \leftarrow \sum_i \mathrm{CE}(\mathrm{LMHead}(H_i), x_{i+1})$;

Backpropagate the final prediction loss and configured balancing loss through all ten passes without detaching any bank;

Table 4: Execution fidelity of the compact study. The same 128 Dev windows and 400M-token checkpoint are evaluated under both graphs. Gaps use full-precision scores; displayed PPL is rounded. Dashes indicate unavailable independent training-sample measurements.

Context and bank	Train PPL	Parallel Dev	Exact Dev	$\lvert\Delta_{\mathrm{graph}}\rvert$
Same / shared	—	33.83	33.83	0.00009
Full / shared	—	31.12	31.28	0.00521
Same / depth-specific	—	32.30	32.31	0.00011
Full / depth-specific	—	29.90	29.97	0.00230
Selective $W = 32$ / shared	—	31.23	31.45	0.00716
Selective $W = 16$ / shared	—	31.60	31.71	0.00350
Selective $W = 8$ / shared	—	31.06	31.17	0.00333
Selective $W = 32$ / depth-specific	—	30.21	30.23	0.00068
Selective $W = 16$ / depth-specific	—	30.18	30.19	0.00033
Selective $W = 8$ / depth-specific	—	30.40	30.44	0.00128

Let $\mathrm{Scan}_\theta(Z, U)$ run all recurrent steps for tokens in U using causal historical reads from Z. It returns their terminal states H and a new bank formed by replacing only their eligible historical states; unselected entries preserve their values and gradient paths. Within the scan, each token reads its own prefix, which it computed earlier in the same pass.

Exact inference. Exact execution replaces the estimated bank with completed past-token states. It processes token i through all R updates, emits its next-token logits, retains the specified history, and only then starts token $i + 1$. Reading a deeper state of an earlier token is causal. Every update reads only past tokens and earlier states of the current token. Independent sequences can be batched, while positions within a sequence stay in token-major order.

B.2 Paired execution-graph evaluation

For a fixed checkpoint and the same evaluation positions, define

$$\Delta_{\mathrm{graph}} = \mathrm{CE}_{\mathrm{exact}} - \mathrm{CE}_{\mathrm{parallel}} . \tag{10}$$

Table 4 reports the absolute difference between mean cross-entropies, in nats per token, computed as $|\log(\mathrm{PPL}_{\mathrm{exact}}/\mathrm{PPL}_{\mathrm{parallel}})|$. Opposite position-wise errors can cancel in this mean, and the aggregate endpoint records omit position-wise losses and logit divergence.

The mean-loss discrepancies range from 0.00009 to 0.00716 nats per token; the largest relative PPL discrepancy is 0.72%. Both the cross-depth gains and the depth-specific-bank advantage retain their direction under exact evaluation. The selective models also remain close to their full-context control. These are fixed-checkpoint checks; assessing convergence would need additional-sweep evaluation and matched retraining. The two-sweep memory study needs its own graph check.

C Additional Controls and Interpretation

C.1 The Single-Layer intervention

Single-Layer refers to a shared attention interface; every control keeps its recurrent computation and its own FFN schedule. A depth-specific expert-bank control still uses that same attention module.

Table 5: Controls attached to each architectural claim. Outcomes appear in the main-text displays.

Question	Held fixed	Relevant checks
Cross-depth access	FFN schedule, R, width, data, optimizer	Extra attention reads, exact quality, graph gap
Expert reuse	Attention rule, resident pool, expert width, within-study routing	Routing, learning curves, active union
Auxiliary input	Base topology and refinement	Added capacity, source information, gradient route
Retained depth	Window, retained-state count, training graph	Exact quality by depth and token age
Practical efficiency	Hardware, precision, batch and context	Latency, throughput, persistent and peak memory

These controlled style variants, inspired by UT, Ouro and Huginn, isolate choices on a common model; comparisons with the original published systems are outside their scope.

The two axes can be intervened on independently, while their effects may interact. Attention itself also transforms representations. Our refinement intervention is specifically the FFN-bank assignment, while communication varies historical access and directed side inputs. This operational definition attributes each difference to a specified architectural change.

Let $L(c, f)$ be exact held-out cross-entropy for communication c and refinement f. The conditional communication effect, holding refinement f fixed, is

$$\delta_{\text{comm}}(f) = L(\text{same}, f) - L(\text{full}, f). \tag{11}$$

For schedules f_1, f_2, $\delta_{\text{comm}}(f_1) - \delta_{\text{comm}}(f_2)$ measures their interaction on the loss scale. A rank reversal is one manifestation; percentage PPL reductions are computed within matched pairs.

C.2 Information, capacity and optimization in side inputs

For start injection, the central contrast is its on/off effect in each base context. A smaller effect under full access supports conditional substitution between the two sources. Injection supplies a current-token prelude state, while cross-token attention exposes previous tokens. Both can change how much the receiving core must reconstruct on its own.

Three further interventions, outside the reported comparisons, could distinguish these explanations. A learned write with a source shuffled among unrelated sequences preserves the module and its parameter count while disrupting token-specific information. Shuffling must not draw later positions from the same target sequence, which would introduce future-token leakage. A stop-gradient source preserves the forward input while removing the direct backward route through that source; indirect gradient paths remain. A same-budget no-input module tests additional transformation capacity. These interventions would be needed for a causal attribution to source information alone. The current result establishes an interaction between injection and historical access. Deep feedback is treated separately under each base topology, as its source and receiving update differ from start injection's.

C.3 Fairness and evidential scope

Equal training tokens and resident capacity establish a controlled architectural comparison; training FLOPs still differ across policies. Broader attention performs extra work, and the training approximation can affect different graphs differently. The method comparison reports analytic state counts alongside exact quality; a compute-matched result requires additional FLOP measurements. A fixed-token result and a compute-matched result answer different questions.

The decomposition stays informative whichever schedule is best. The value of a shared bank is assessed against scheduled banks at the same capacity. Showing that an earlier UT-style result was undertrained would require learning curves and a training-budget comparison.

Historical redundancy is an empirical quality–cost result: a subset must preserve measured capability while removing states or reads. The axis and boundary studies establish the observations in their respective larger-capacity settings. The focused 20M comparison tests the within-setting communication and refinement effects under a compact configuration. Recurrent depth, expert width, routing and training schedule also differ across the two settings, so capacity scaling remains open.

D Communication and Compute Accounting

D.1 Persistent history versus attention reads

With d_{L} and d_{W} distinct retained depths for old and recent tokens, the state count after T tokens is

$$N_{\mathrm{sel}}(T) = (T-W)_+ d_{\mathrm{L}} + \min(T,W) d_{\mathrm{W}}. \tag{12}$$

For one long-term anchor and a complete recent trajectory, $N_{\mathrm{sel}} = T + (R-1)\min(T,W)$. With H_{KV} K/V heads, head dimension d_h and b bytes per scalar, the projected historical-state payload is

$$M_{\mathrm{history}} = 2bH_{\mathrm{KV}} d_h N_{\mathrm{sel}}(T). \tag{13}$$

The factor two accounts for keys and values. This excludes allocator overhead, metadata, current-token states and training activations. At $R = 8, T = 256, W = 8$, the analytic payload is 312 records versus 2,048 for full cross-depth persistence. In BF16 with four 64-dimensional K/V heads these are 0.305 and 2.000 MiB per sequence. The reduction is 84.8% in this payload, not total device memory.

Same-depth and full cross-depth can require the same TR resident records under exact token-major execution, while reading different numbers of records at each update. Window-only stores $R\min(T,W)$ records; anchor-only stores T. A selected anchor remains for every historical token, so long-term storage grows linearly with context length.

D.2 FLOPs and runtime

For a query reading n historical/current states, the dense $QK^\top$ and attention–value products contribute approximately $4H_Q d_h n$ FLOPs, with one multiply-add counted as two operations. Grouped-query attention reduces stored K/V width while query-head attention products remain. Query, key, value and output projections, normalization, gating, routing, activated FFNs and the language-model head are counted separately. A complete original token includes all R updates.

Training cost sums actual forward, backward and recomputation work:

$$F_{\mathrm{train}} = \sum_{s\in\mathcal{F}} F^{(s)}_{\mathrm{fwd}} + \sum_{s\in\mathcal{G}} F^{(s)}_{\mathrm{bwd}} + F_{\mathrm{recompute}} + F_{\mathrm{loss}}. \tag{14}$$

Here $\mathcal{F}$ contains executed scans and $\mathcal{G}$ their gradient-tracked subset. The compact configuration has ten passes in each set: two full-token passes and eight half-token parity passes. These cover six full-token-scan equivalents when the two parities have equal size, so the ten gradient-tracked passes amount to six full-scan backward workloads. Actual tensor shapes, the initialization graph and readout work enter the FLOP estimator separately. The larger-model studies retain their own schedules. An implementation-level estimate must distinguish logical operations under causal masks from operations executed by padded kernels. Table 2 reports analytic matrix-product counts for the shared-bank strict decoder, averaged over 256 token positions. They include projections, activated experts, router, attention products and the final language-model head, but exclude normalization, softmax, RoPE, memory transfers and kernel overhead. The current same-depth kernel computes dense attention before masking, giving the same count as full cross-depth access. These are implementation-specific estimates; an optimized same-depth kernel could cost less.

Measured latency, throughput and peak device memory are outside the reported compact comparison. Such a comparison must hold hardware, precision, batch and context fixed, record warmup and synchronization, and report repeated timings in original tokens per second. Maximum-fitting-batch throughput and fixed-batch speedup answer different questions. Training peak memory also includes the source/destination banks and retained activations of every pass.

E Graph Discrepancy and Diagnostics

E.1 Dependency propagation

The exact graph is a causal token–depth DAG. Under a deterministic update and complete within-token depth scans, Jacobi sweeps propagate a correct prefix: once all predecessor-token states are exact, the next token is exact on its next sweep. Starting with arbitrary historical estimates, the first k

Table 6: Which depths should remain readable? Exact Dev PPL for the larger-capacity memory study, $R = 16$, context 256, 400M training tokens and $W = 32$. Each slice varies only the added boundaries in the named region; the complementary region keeps its segment-entry path. All denotes $\{z_0, z_8, z_{16}\}$. The no-extra-anchor reference uses a different training graph and is separated from the matched anchor additions. Bold marks the lowest-PPL anchor addition in each column.

	Distant-history intervention		Recent-history intervention	
Added states	Shared	Depth-specific	Shared	Depth-specific
None (reference)	31.96	23.86	31.96	23.86
z_0	**28.28**	**23.78**	**28.08**	23.74
z_8	48.14	24.37	35.02	**23.25**
z_{16}	35.90	28.65	40.12	31.23
All	53.96	27.27	35.50	25.68

tokens are therefore exact after at most k full sweeps (up to the total sequence length). This induction gives a finite worst-case bound for full Jacobi sweeps, far above the four alternating rounds used for a 256-token context. Useful approximate agreement can occur earlier and must be measured.

For a local error analysis, let $F_{\theta,j}(h, \mathcal{M})$ denote one update. Assume, over the compared states and a fixed routing region in which expert selections stay constant, that

$$\|F_{\theta,j}(h, \mathcal{M}) - F_{\theta,j}(h', \mathcal{M}')\|_2 \le L_h \|h - h'\|_2 + L_M d(\mathcal{M}, \mathcal{M}'). \tag{15}$$

Writing $e_{i,j} = \|h_{i,j} - \tilde{h}_{i,j}\|_2$ and using the same embedding in both executions yields

$$e_{i,R} \le L_M \sum_{j=0}^{R-1} L_h^{R-1-j} d(\mathcal{M}_{i,j}, \tilde{\mathcal{M}}_{i,j}). \tag{16}$$

If projected states are aligned and their projection is L_{KV}-Lipschitz, a maximum-slot discrepancy satisfies $d \le L_{KV} \max_{(m,n)\in\mathcal{C}_{i,j}} e_{m,n}$. Hard expert selection can change operators at a boundary, so these assumptions hold only locally, within a fixed routing region.

These expressions identify error paths. A long-lived anchor can expose the same write error to many later tokens; a working-memory state eventually leaves the window. State error, attention-output sensitivity and logit divergence identify which discrepancies affect prediction. A large hidden-state error can still produce a small loss error when the affected states receive little attention.

E.2 Learning dynamics and robustness

The reported compact diagnostics are paired parallel/exact endpoint scores. Exact train/Dev curves at matched token checkpoints, expert utilization by depth, and paired seed differences would resolve additional questions about optimization and robustness beyond these endpoints. Confidence intervals over independently trained seeds characterize training variability; token-level resampling captures only evaluation noise. Window-level uncertainty for a fixed checkpoint is labeled separately. The single-run injection and feedback differences have no estimated between-seed uncertainty, so near-zero differences, such as those for deep feedback, remain inconclusive.

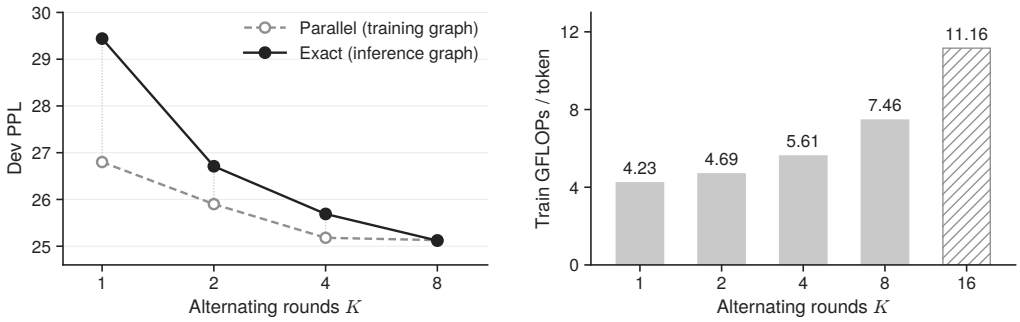
F Architecture-Study Contracts and Update Pilot

This section documents the configurations underlying the first two main-text studies and the earlier update-count pilot. They are distinct from the compact-model validation and from one another.

F.1 Communication–refinement matrix

The axis matrix uses $R = 16$, context 256, width 512, eight query heads, four K/V heads, and a 16K vocabulary. Its 48 routed experts have width 3,600, allocated as 1×48, 2×24, 3×16 or 16×3. Each step activates one always-on shared expert and one routed expert. Resident capacity is approximately 274.42–274.43M, whereas per-step active capacity is approximately 20.04–20.07M. These are different quantities. Runs use 450,035,712 tokens and K8 partitioned updates, differentiating the last two even/odd rounds and readout scan (five gradient-tracked scans).

Table 1 reports this matrix, including its start-injection and deep-feedback variants.



(a) Execution agreement **(b) Training cost**

Figure 6: Execution agreement improves at an increasing training cost in the update pilot. (a) Parallel and exact Dev PPL from the same checkpoint for each update count. (b) Estimated training work includes forward and backward FLOPs. Each endpoint is from a separately trained 400M-token run at $R = 16$, $W = 32$, anchor 2, and context 4096. The $K = 1$ pilot tracks four gradient scans; the remaining settings track five. $K = 16$ (hatched) appears only in panel (b), as a compute estimate.

Table 7: Earlier alternating-update pilot. These counts differ from the ten full-gradient passes of the compact four-round configuration. FLOPs include forward and backward work. The $K = 16$ PPL endpoint is unavailable.

K	Forward scans	Gradient scans	Training cost GFLOPs per token	Parallel Dev PPL	Exact Dev PPL
1	4	4	4.23	26.80	29.44
2	6	5	4.69	25.90	26.71
4	10	5	5.61	25.18	25.69
8	18	5	7.46	25.13	25.12
16	34	5	11.16	—	—

F.2 Boundary-state selection

The memory study uses $R = 16$, $W = 32$, context 256 and boundaries $\{0, 8, 16\}$. A query-matched segment-entry path remains available. One slice adds anchors outside the working window; the other adds them inside it. Anchor variants use two eight-step segments, two direct-overwrite block-Jacobi sweeps, and gradients through the seed and both sweeps. The no-extra-anchor reference is trained on its exact graph, so differences from that reference also include a training-graph difference. Anchor-to-anchor comparisons share their training contract.

The shared model has 274,429,568 resident parameters and uses the larger expert configuration above. Training uses 400,031,744 tokens, global batch 128, microbatch 32, seed 42 and the compact study's AdamW schedule. The model runs without injection, feedback or depth modulation. The depth-specific endpoints form a separate slice, and policies are compared within each slice.

The measured endpoints are in Table 6. Their graph discrepancy must be checked within this schedule, separately from the alternating-update pilot below. Paired parallel/exact scores for these boundary variants, including the all-boundary condition, are unavailable in the current endpoint record. The result therefore describes the exact quality of the trained variants under this schedule.

F.3 Alternating-update pilot

The update-count pilot fixes $R = 16$, $W = 32$, $a = 2$, context 4096 and an approximately 97.48M resident model. It uses a 16-expert, width-3,600 configuration with depth-specific normalization gains, and trains for 400,031,744 tokens. Evaluation uses 128 fixed 4096-token Dev windows. The initial scan and all but the last two alternating rounds are detached for $K \geq 2$; the final two rounds and readout carry gradients. The $K = 1$ pilot differentiates all four configured scans. Each half-batch scan counts as a pass, with its actual work counted in FLOPs.

Figure 6 separates the quality gap from its compute cost. Table 7 lists all pass counts. Agreement at $K = 8$ is evidence about this pilot. The compact model has its own paired check in Table 4.