

Recurrent Looped Transformer

Yifan Zhang¹ Jichen Feng² Shihan Qin²
 Sun-Yuan Kung¹ Mengdi Wang¹

¹Princeton University ²University of Pennsylvania

Abstract

State tracking requires an update at every input, but the depth a Transformer applies to each token is fixed regardless of sequence length. We introduce the Recurrent Looped Transformer (RLT), which splits its layers between a parallel causal encoder and a recurrent decoder. At each token, the decoder merges the encoder output with the previous token’s final decoder state, so the computation path grows with sequence length at a fixed per-token cost. On six algorithmic tasks, we compare five splits of eight layers with an eight-layer Transformer over three seeds. Trained on at most 40 bits, two RLT splits generalize parity to 256 bits with 100% accuracy in every seed, while the Transformer stays at chance. On swap-based S_5 permutation tracking at eight times the training length, RLT reaches 97% final-state accuracy versus under 1% for the Transformer, and accuracy increases with decoder depth. On modular arithmetic beyond the training lengths, RLT reaches up to 93% versus 33% for the Transformer. Ablations show that these gains depend on the feedback: removing it drops parity and swap-based S_5 to chance at every split. Updating the feedback once per four-token chunk lets known tokens in a chunk run in parallel and keeps 64-bit parity at 99%, while permutation tracking depends on per-token feedback: chunking lowers length-64 swap-based S_5 from 100% to 20%.

Project Page: <https://github.com/yifanzhang-pro/recurrent-looped-tranformer>

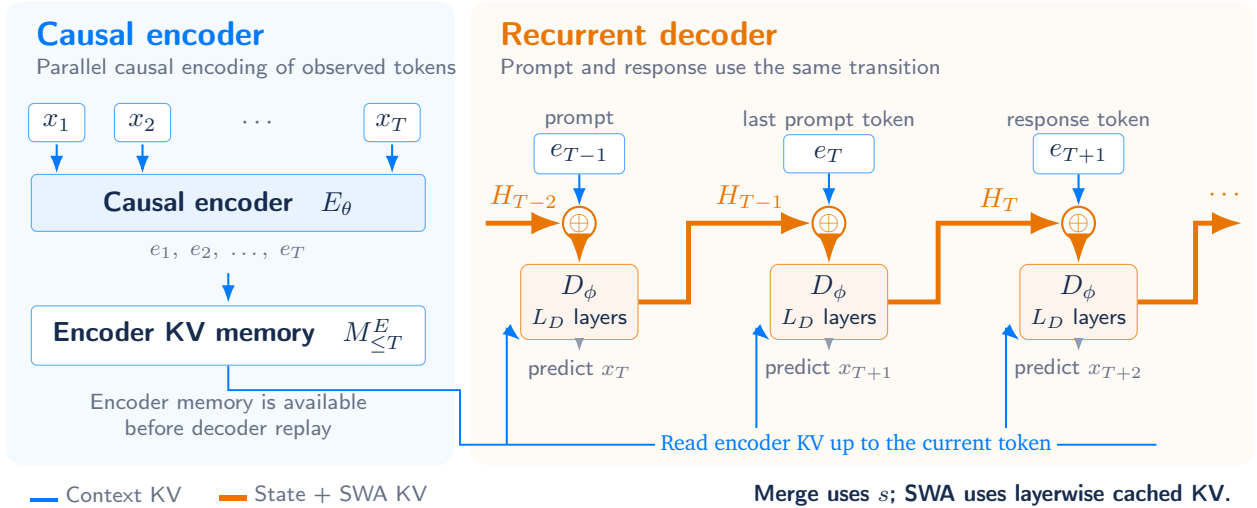


Figure 1 Recurrence across prompt and response. The encoder builds causal key-value (KV) memory; the decoder processes tokens in order. Orange arrows carry the decoder state $H_t = (s_t, C_t^D)$: the merge reads s_t , and each SWA layer reads its own cache. Blue arrows supply encoder memory up to the current position. The drawing shows the last two prompt updates and the first response update.

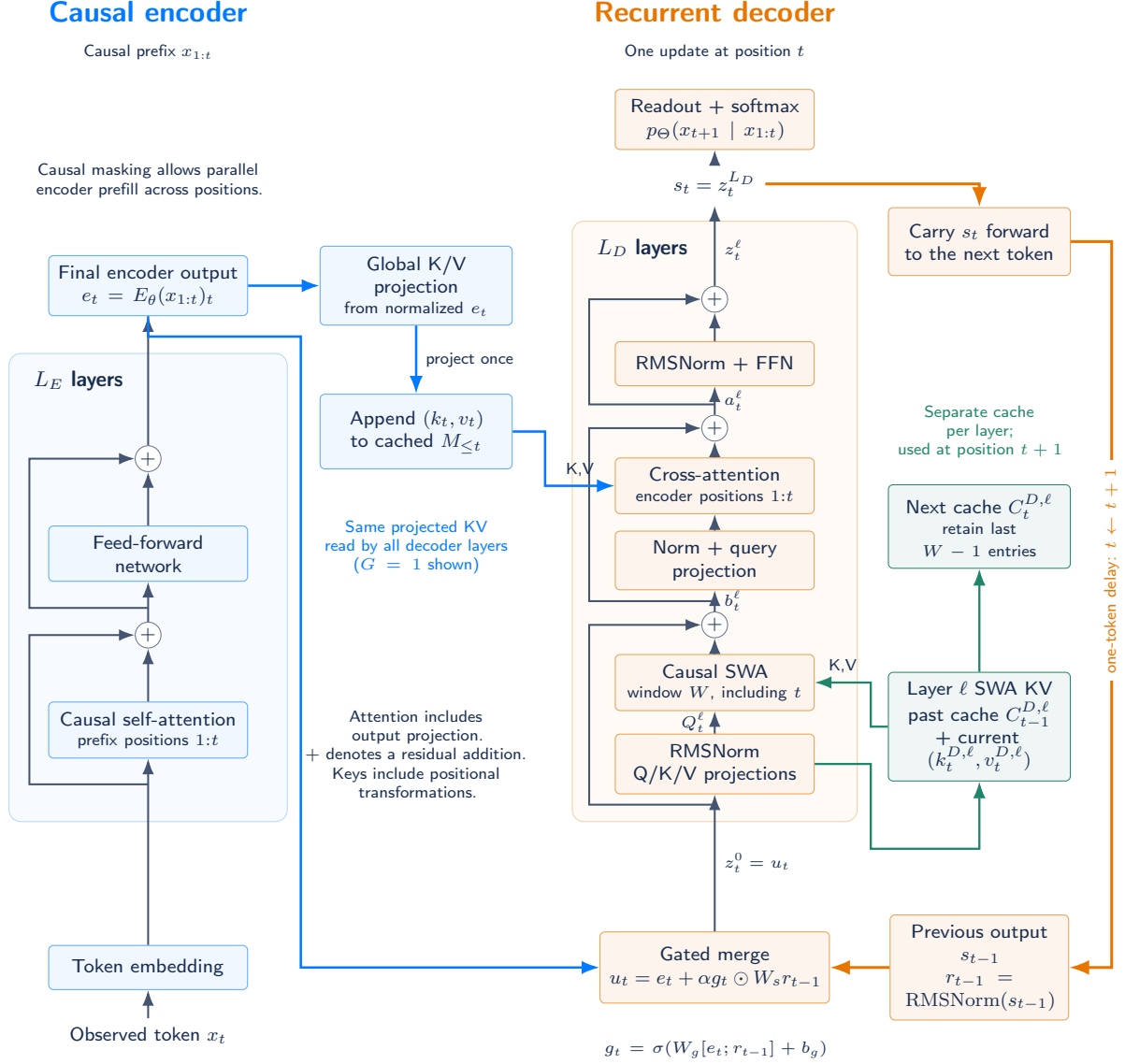


Figure 2 RLT-1 architecture for one token update. Computation flows upward; each outlined stack repeats for the indicated depth. The encoder output supplies both the gated merge and the global KV projection. With $G = 1$, all decoder layers use their own queries to read the same cached, projected global KV. Each decoder layer constructs separate SWA KV from its own input, attends to its local window, then reads global memory and applies an FFN. Residual paths bypass each attention or FFN sublayer. The final decoder output s_t feeds the next token’s merge, while each layer retains its own SWA cache. Blue arrows carry encoder features and global memory, green arrows carry local KV, and orange arrows carry recurrent hidden states.

1 Introduction

Algorithmic state tracking requires a model to update a state with each input: parity accumulates bits, and permutation tracking composes group elements. A Transformer computes each position with a fixed number of layers, and unless $\text{TC}^0 = \text{NC}^1$, fixed-depth Transformers cannot track compositions of S_5 permutations over arbitrarily long sequences (Merrill et al., 2024). A recurrent network instead passes the result of each update to the next, so its computation path grows with the input. Adding this connection to a Transformer lets later tokens build on earlier computation along a path that grows with sequence length.

We introduce the Recurrent Looped Transformer (RLT), which divides its layers between a causal encoder and a recurrent Transformer decoder (Figure 1). The encoder processes known tokens in parallel and builds token representations and global key-value memory. At each position, a gated merge combines the current encoder representation with the previous token’s final decoder output. The decoder then attends to the encoder prefix and its own recent activations before predicting the next token. Prompt and response tokens follow the same transition. With decoder depth L_D , processing t tokens creates a recurrent path through tL_D decoder blocks, while each token passes through only $L_E + L_D$ layers.

This design introduces two choices that a decoder-only Transformer lacks. The first is the split between encoder and decoder depth: at a fixed total layer count, moving layers to the decoder lengthens the recurrent path but adds decoder work that follows token order during training and prefill. The second is the feedback interval B , the number of tokens between feedback updates. RLT-1 feeds back at every token ($B = 1$), RLT-2 shares one feedback state across each chunk of B tokens, and RLT-0 removes the feedback ($B = \infty$). For T known tokens, the decoder needs TL_D sequential block stages in RLT-1, $\lceil T/B \rceil L_D$ in RLT-2, and L_D in RLT-0 (Table 1); RLT-2 uses the same parameters as RLT-1.

We compare five allocations of eight layers with an eight-layer decoder-only Transformer on six algorithmic tasks: addition, parity, modular arithmetic with and without brackets, and standard and swap-based S_5 state tracking. All comparisons use three initialization seeds and shared training and test data, and evaluate lengths well beyond the training range. Our contributions are as follows.

- **Architecture.** RLT combines encoder-memory reuse (Sun et al., 2024) with feedback through the full decoder at every token. The appendices specify training by backpropagation through the full recurrent history, cache reuse across turns, and exact policy replay.
- **Length generalization.** Trained on at most 40 bits, RLT splits $5 + 3$ and $7 + 1$ reach $100 \pm 0\%$ parity accuracy at 256 bits in every seed, compared with $50.07 \pm 1.63\%$ for the Transformer. On swap-based S_5 at 256 operations, eight times the training length, $4 + 4$ reaches $97.30 \pm 2.76\%$ final-state accuracy, compared with $0.85 \pm 0.30\%$. After 5,000 training steps, $6 + 2$ reaches $93.36 \pm 5.69\%$ on flat mod-5 expressions of length 63, compared with $33.20 \pm 2.33\%$ (Section 3.2).
- **Depth allocation.** The best split depends on the task: parity generalizes best with one or three decoder layers, whereas swap-based S_5 accuracy at 256 operations rises with decoder depth, from chance with one decoder layer to 97% with four.
- **Feedback interval.** Without feedback, parity and swap-based S_5 stay near chance from length 64 at every split. With four-token chunks, RLT-2 keeps 64-bit parity at $98.99 \pm 1.66\%$ but lowers swap-based S_5 at 64 operations from 100% to $19.60 \pm 6.54\%$, so permutation tracking benefits most from per-token feedback.

Feedback Transformer, Recurrent Transformer, Full-bandwidth Transformer, T²MLR, and Latent Recurrent Transformer also pass information across tokens through feedback (Fan et al., 2020; Oncescu et al., 2026; Wang et al., 2026; Cai et al., 2026; Huang et al., 2026). RLT feeds back through the full decoder, keeps this recurrence separate from parallel encoder memory, and treats the allocation of depth between encoding and recurrent updates as a design variable (Section 4).

2 Method

This section defines RLT-1 and two controls that change its feedback interval.

2.1 Sequence and state

Let $x_{1:S}$ be an independent sequence beginning with $x_1 = \text{BOS}$. At inference, $x_{1:T}$ is the prompt and later tokens form the response. The same conditional distribution applies before and after T . Let L_E and L_D be the encoder and decoder depths, and let d be the residual width. We write the encoder representation $e_t \in \mathbb{R}^d$ and recurrent output $s_t \in \mathbb{R}^d$ as column vectors.

The complete decoder state is $H_t = (s_t, C_t^D)$. The cache C_t^D holds the key/value projections retained by each decoder SWA layer. The window size $W \geq 1$ includes the current token, so each layer retains at most $W - 1$ past positions after an update. We collect all parameters, including the learned initial state s_* , in Θ . The encoder and decoder are denoted by E_θ and D_ϕ .

2.2 Causal encoder and memory

For an observed prefix, compute

$$e_{1:T} = E_\theta(x_{1:T}). \quad (2.1)$$

Positions within each encoder layer can be processed together using a causal mask. For decoder memory group $g \in \{1, \dots, G\}$, construct

$$k_t^g = \mathcal{P}_K^g(e_t, t), \quad v_t^g = W_V^g \text{RMSNorm}_E(e_t), \quad M_{\leq t}^g = \{(k_j^g, v_j^g)\}_{j=1}^t. \quad (2.2)$$

The key map includes normalization, projection, and any positional transformation. Decoder layer ℓ reads group $g(\ell)$: $G = 1$ shares memory across layers, while $G = L_D$ allows separate projections for each layer. Separate attention sublayers read the two stores: cross-attention reads encoder memory $M_{\leq t}$ over the full prefix, and causal SWA reads decoder KV within a bounded window.

2.3 One transition for every token

Initialize once, before BOS:

$$H_0 = (s_*, \emptyset). \quad (2.3)$$

For every observed or sampled token, apply

$$u_t = \text{Merge}(e_t, s_{t-1}), \quad (2.4)$$

$$H_t = (s_t, C_t^D) = D_\phi(u_t; M_{\leq t}, C_{t-1}^D, t), \quad t \geq 1, \quad (2.5)$$

$$p_\Theta(x_{t+1} \mid x_{1:t}) = \text{softmax}(W_o \text{RMSNorm}_o(s_t))_{x_{t+1}}. \quad (2.6)$$

Define $F_t(H) = D_\phi(\text{Merge}(e_t, s); M_{\leq t}, C^D, t)$ for $H = (s, C^D)$. After the encoder features are computed, the decoder processes every prompt position to predict the first response token:

$$H_T = F_T \circ F_{T-1} \circ \dots \circ F_1(H_0). \quad (2.7)$$

Generation continues from both components of H_T . After sampling x_{T+1} from s_T , encode it incrementally, append its encoder-derived KV, and compute $H_{T+1} = F_{T+1}(H_T)$, including every decoder SWA cache update.

RNN and state-space interpretations. RLT-1 can be viewed as a recurrent neural network (RNN) whose cell is the gated merge followed by the full Transformer decoder. The cell reuses its parameters across tokens, while its decoder layers may have distinct parameters. For a decoder state $H = (s, C^D)$, encoder representation e , and encoder memory M , define the state-transition function and output function by

$$\mathcal{F}_\Theta(H; e, M, t) := D_\phi(\text{Merge}(e, s); M, C^D, t), \quad (2.8)$$

$$\mathcal{G}_\Theta(H) := \text{softmax}(W_o \text{RMSNorm}_o(s)). \quad (2.9)$$

The nonlinear state-space equations are then

$$H_t = \mathcal{F}_\Theta(H_{t-1}; e_t, M_{\leq t}, t), \quad (2.10)$$

$$p_\Theta(\cdot \mid x_{1:t}) = \mathcal{G}_\Theta(H_t). \quad (2.11)$$

The transition updates both the feedback vector s_t and the decoder cache C_t^D ; the output function maps s_t to a distribution over the next token. The encoder supplies e_t and $M_{\leq t}$ as conditioning inputs, so the earlier notation is $F_t(H) = \mathcal{F}_\Theta(H; e_t, M_{\leq t}, t)$. At fixed width, decoder depth, and SWA window, this decoder state has bounded size, while the encoder memory $M_{\leq t}$ grows with sequence length.

The state-space model (SSM) cores of S4 and Mamba use updates that are affine in the previous state once the layer inputs are fixed; Mamba makes the coefficients input-dependent (Gu et al., 2021; Gu and Dao, 2023). S4 exploits time invariance for convolutional evaluation, while Mamba uses parallel scans of its input-dependent affine updates. RLT-1’s gated merge and decoder instead depend nonlinearly on the previous decoder state, so these affine scan algorithms do not directly apply to its transition.

2.4 Prompt prefix and incremental decoding

Prefill first computes the causal encoder representations and memory of the prompt, then evaluates $H_1, \dots, H_T$ in order. At decoder position t , attention is restricted to $M_{\leq t}$ even though the entire prompt memory is available. Generation uses

$$(e_t, C_t^E) = E_\theta^{\text{step}}(x_t, C_{t-1}^E) \quad (2.12)$$

with encoder cache C^E , followed by a memory append and decoder update. Observed tokens may be encoded in chunks if the encoder cache is preserved. The decoder still processes every token in order: each update produces the hidden state and SWA KV needed by later positions.

2.5 Merge and decoder blocks

We use the following gated merge:

$$r_{t-1} = \text{RMSNorm}_s(s_{t-1}), \quad (2.13)$$

$$g_t = \sigma(W_g[e_t; r_{t-1}] + b_g), \quad (2.14)$$

$$u_t = e_t + \alpha g_t \odot W_s r_{t-1}. \quad (2.15)$$

Here $W_g \in \mathbb{R}^{d \times 2d}$, $W_s \in \mathbb{R}^{d \times d}$, and α controls the feedback scale. The main comparison uses $\alpha = 0.1$.

Starting from $z_t^0 = u_t$, each decoder block applies causal SWA, encoder-memory cross-attention, and a feed-forward network (FFN):

$$q_t^{D,\ell} = \mathcal{P}_Q^{D,\ell}(z_t^{\ell-1}, t), \quad (2.16)$$

$$k_t^{D,\ell} = \mathcal{P}_K^{D,\ell}(z_t^{\ell-1}, t), \quad v_t^{D,\ell} = W_V^{D,\ell} \text{RMSNorm}_{S,\ell}(z_t^{\ell-1}), \quad (2.17)$$

$$b_t^\ell = z_t^{\ell-1} + \text{Attn}_\ell^D \left(q_t^{D,\ell}, \{(k_j^{D,\ell}, v_j^{D,\ell})\}_{j=\max(1, t-W+1)}^t \right), \quad (2.18)$$

$$a_t^\ell = b_t^\ell + \text{Attn}_\ell^M \left(\mathcal{P}_Q^{M,\ell}(b_t^\ell, t), M_{\leq t}^{g(\ell)} \right), \quad (2.19)$$

$$z_t^\ell = a_t^\ell + \text{FFN}_\ell(\text{RMSNorm}_{D,\ell}(a_t^\ell)), \quad s_t = z_t^{L_D}. \quad (2.20)$$

The attention operators include output projections; query/key maps include their normalizations and positional transformations. Each layer forms current KV from its input before SWA, so attention to the current position introduces no circular dependency. Historical decoder KV comes from C_{t-1}^D . After the update, retain positions $\max(1, t - W + 2), \dots, t$ in C_t^D ; this set is empty for $W = 1$. Figure 2 expands the encoder and decoder stacks for one token update.

2.6 Recurrent depth and execution cost

After a prompt of length T and n processed response tokens, the feedback path has passed through $(T + n)L_D$ decoder blocks, while each token evaluates $L_E + L_D$ blocks (Figure 3). The influence of earlier states depends on the gates, projections, and products of transition Jacobians. A prompt of length T requires T sequential decoder updates; updates from independent sequences can share a batch, with separate states and caches. Appendix B derives the work and storage costs and proves that states and next-token distributions are invariant to the prompt–response split.

2.7 Feedback interval: RLT-1, RLT-2, and RLT-0

RLT-1 and its two controls differ in one quantity, the feedback interval B : the number of tokens between updates of the state that enters the decoder input. RLT-1 updates this state at every token ($B = 1$), RLT-2 once per chunk of B tokens, and RLT-0 never ($B = \infty$). All three keep the causal encoder, encoder memory, decoder SWA, prefix-restricted memory attention, and readout of RLT-1.

RLT-2 holds the feedback state fixed within a chunk of B tokens and updates it at the chunk boundary (Figure 4). Boundaries are anchored at BOS, so chunk k covers the known positions $I_k = \{(k-1)B + 1, \dots, \min(kB, T)\}$. Starting from the feedback register $h_0 = s_*$, every position in

Block count per token and along the recurrent path

Illustrative 48-layer encoder + 48-layer decoder; counts exclude merge and readout.

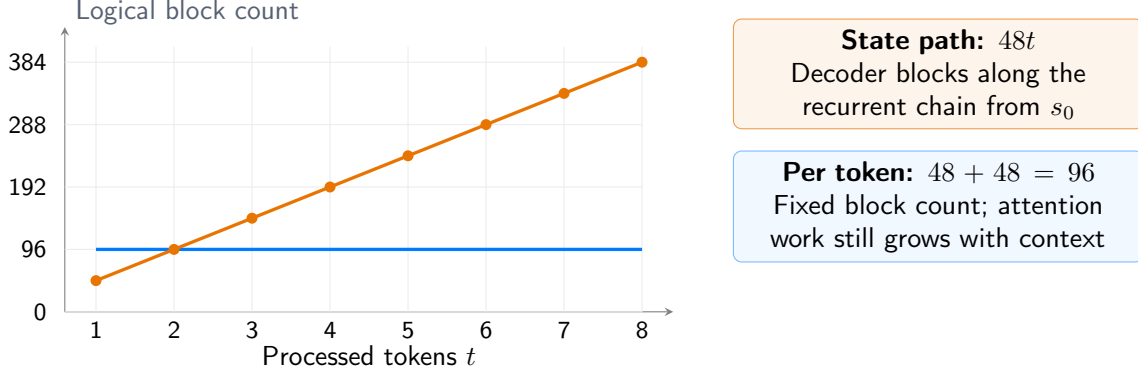


Figure 3 Recurrent path length and block count per token. For $L_E = L_D = 48$, the path traverses $48t$ decoder blocks after t tokens. Each token evaluates 96 encoder and decoder blocks. Orange counts decoder blocks along the recurrent path; blue counts blocks in both stacks per token.

chunk k merges its encoder output with the same boundary state through its own gate:

$$r_{k-1} = \text{RMSNorm}_s(h_{k-1}), \quad g_t = \sigma(W_g[e_t; r_{k-1}] + b_g), \quad (2.21)$$

$$z_t^0 = e_t + \alpha g_t \odot W_s r_{k-1}, \quad t \in I_k, \quad (2.22)$$

$$h_k = y_{kB} \quad \text{when } kB \leq T. \quad (2.23)$$

Here $y_t = z_t^{L_D}$ is the final decoder output at position t ; every y_t predicts x_{t+1} , but only the last output of a complete chunk becomes the next feedback state. The decoder applies Equation (2.20) to all known positions of a chunk together, and causal SWA also reads keys retained from earlier chunks. Because h_{k-1} depends only on tokens before the chunk, the causal masks keep each y_t a function of $x_{1:t}$. A partial final chunk leaves the register at the preceding complete boundary, so moving the prompt–response split does not change the computation. With $B = 1$, every token closes a chunk, and RLT-2 reduces exactly to RLT-1.

RLT-0 is the $B = \infty$ limit: no chunk boundary is reached, so no decoder output is fed back. RLT-2 with $B = \infty$ would still merge every position with the constant initial state s_* ; RLT-0 drops this constant merge together with s_* , state normalization, the gate, and the feedback projection, and feeds $z_t^0 = e_t$ to the decoder (Figure 14).

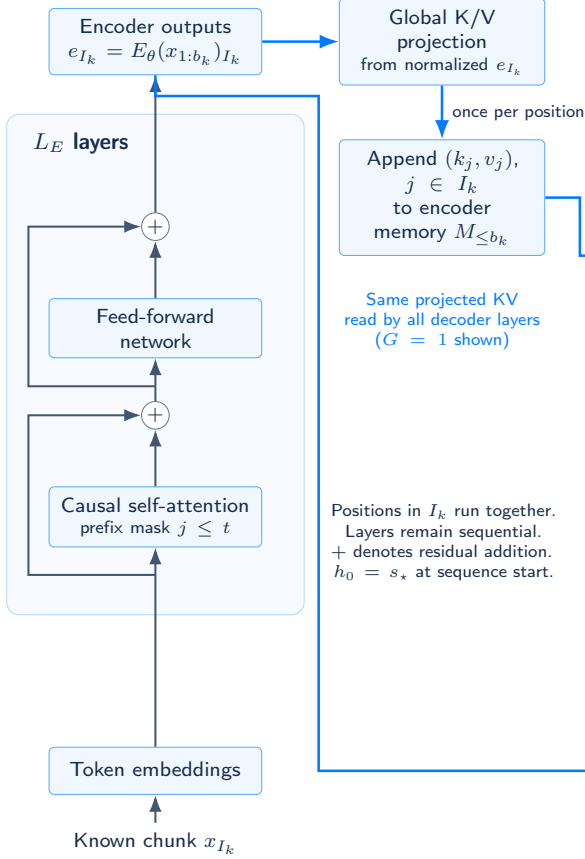
Table 1 and Figure 5 compare the three intervals at matched stack dimensions. For a known prefix of length T , the decoder needs $\lceil T/B \rceil L_D$ sequential block stages, with up to $\min(B, T)$ positions in each batched matrix operation: TL_D stages for RLT-1 and L_D for RLT-0. All three evaluate TL_D decoder blocks and generate one token per step. Larger B therefore trades feedback frequency for known-token parallelism. Appendix I gives the chunked training, prefill, and incremental-generation procedures.

Because RLT-2 uses the same parameters for every B , the chunk size can also vary during training. Larger chunks shorten the sequential decoder path, while smaller chunks extrapolate better on parity and swaps- S_5 (Section 3). In seed-42 CPU timing at $4 + 4$, chunk4 training steps run 2.27 times as fast as RLT-1 steps, and RLT-0 steps 4.17–4.30 times as fast (Appendix J.10). Pretraining can

Causal encoder

Known prefix through b_k

Causal masking allows parallel encoder prefill across positions.



RLT-2 decoder

Chunk I_k : parallel positions within each layer

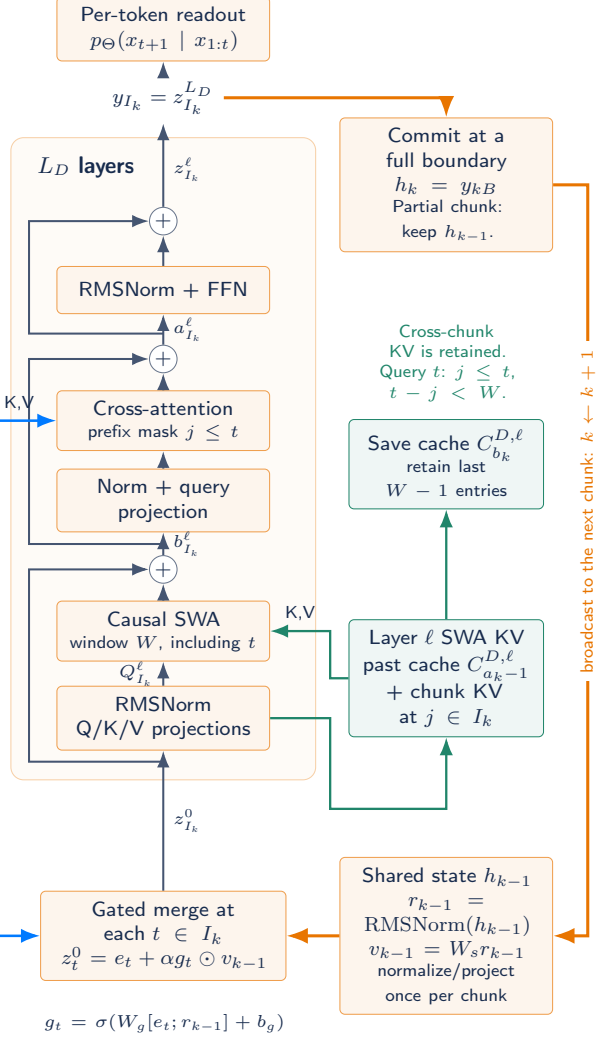


Figure 4 RLT-2 architecture for one chunk. Computation flows upward; the stacks repeat for the indicated depths. Each position $t \in I_k$ uses its own gate to merge its encoder output with the preceding boundary state h_{k-1} . State normalization and projection are shared across the chunk. Each decoder layer applies causal SWA, prefix-restricted encoder-memory attention, and an FFN, with residual connections around each sublayer. For $G = 1$, all decoder layers read the same projected encoder KV; each layer retains its own SWA KV across chunk boundaries. Every output y_t predicts the next token; only y_{kB} at a complete boundary becomes the next feedback state. A partial chunk preserves h_{k-1} during continuation. Blue arrows carry encoder features and memory, green arrows carry decoder KV, and orange arrows carry chunk-level feedback.

Quantity	RLT-1 ($B = 1$)	RLT-2 (B)	RLT-0 ($B = \infty$)
Positions per decoder batch, known tokens	1	up to B	T
Decoder block stages, training forward / prefill	TL_D	KL_D	L_D
Backward dependency depth, full BPTT	$O(TL_D)$	$O(KL_D)$	$O(L_D)$
Final-state feedback updates over T tokens	T	$\lceil T/B \rceil$	0
Decoder block evaluations over T tokens	TL_D	TL_D	TL_D
Merge gate evaluations over T tokens	T	T	0
Shared feedback projections, known tokens	T	K	0
Autoregressive token steps for N tokens	N	N	N
Blocks per consumed generation token	$L_E + L_D$	$L_E + L_D$	$L_E + L_D$
Persistent feedback register	d values	d values	none

Table 1 Execution costs at matched stack dimensions. T counts known input tokens, $K = \lceil T/B \rceil$, and N counts generated tokens; all three variants share the L_E -stage causal encoder prefill. Counts omit common readout and encoder-memory projection work. RLT-2 reuses the normalized and projected boundary state; gates depend on each token. Backward counts cover the decoder dependency graph, with the encoder’s backward pass common to all variants. Counts describe dependencies and arithmetic; wall-clock speed depends on the implementation and hardware.

therefore begin with a large B , processing most tokens with high parallelism, and mid-training and post-training can reduce B toward $B = 1$ to adapt the model to frequent feedback. Changing B changes the model’s computation, so inference and RL replay use the chunk size of the final training stage. Our experiments train each chunk size from scratch and do not test such a schedule.

2.8 Training and state replay

Training uses teacher forcing and cross-entropy on the selected next-token targets. Full backpropagation through time (BPTT) follows recurrent outputs, decoder KV, and encoder memory; masking a target loss does not skip its state update. For reinforcement learning (RL), evaluating a sampled response after a parameter update requires replaying its history to rebuild the state under the current parameters. Appendix D gives the objectives and replay conditions for pretraining, supervised fine-tuning, and policy gradients.

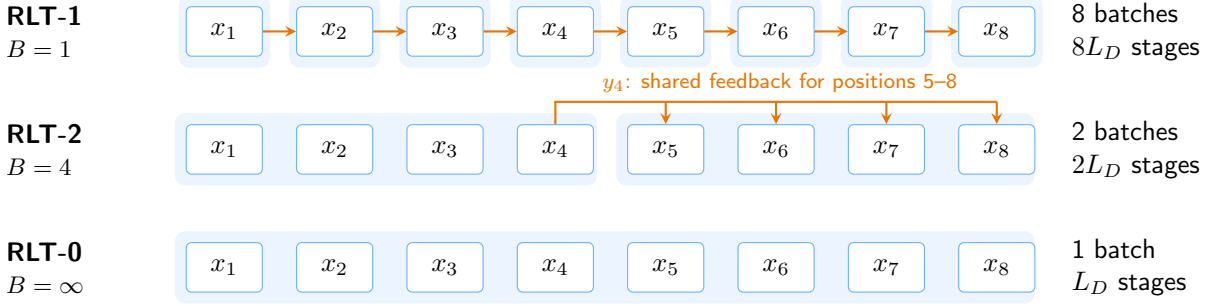
3 Algorithmic Experiments

We study how decoder depth and feedback frequency affect length generalization. The experiments cover addition, parity, modular arithmetic with and without brackets, and standard and swaps-based S_5 state tracking. We first compare encoder–decoder allocations, then vary the feedback interval at a fixed split. Appendix J.1 gives the complete six-task study at 2,000 steps; Appendices J.9 and J.10 give all feedback variants and the 5,000-step modular-arithmetic comparisons.

3.1 Models and evaluation

All models have eight logical layers, width 512, FFN width 1,365, and four attention heads. We compare untied RLT-1 splits $4 + 4$, $5 + 3$, $6 + 2$, $7 + 1$, and $8 + 0$ with an eight-layer decoder-only Transformer (GPT Layer 8). RLT-1 uses an SWA window of eight, one shared encoder-memory group,

Known-token decoder batches



Within a batch: positions in parallel at each layer. Between batches: preserve causal SWA caches.

Figure 5 Decoder scheduling for eight known tokens as the feedback interval grows. Rows show RLT-1 ($B = 1$), RLT-2 ($B = 4$), and RLT-0 ($B = \infty$). Each shaded group is one layerwise decoder batch, containing L_D sequential blocks. Orange arrows show feedback dependencies; RLT-2 broadcasts y_4 to all four positions in its second chunk, and RLT-0 has none. Causal SWA applies in every row, with caches retained between batches. During autoregressive generation, all three variants consume tokens one at a time.

and feedback scale $\alpha = 0.1$. The $8 + 0$ variant retains the gated recurrent merge despite having no decoder blocks. Equal layer counts do not match parameters or compute: RLT-1 has 26.10–28.73M parameters, compared with 25.31M for the Transformer (Table 4).

All eight-layer accuracy comparisons use initialization seeds 42, 43, and 44, with a shared training stream and held-out examples within each task. Global batch size is 512 and microbatch size is 32. AdamW uses $(\beta_1, \beta_2) = (0.9, 0.95)$, weight decay 0.1, and gradient clipping at norm 1. The base learning rate warms up to 10^{-4} over 200 steps, then decays to 5×10^{-6} at the end of each run. Parity, addition, and S_5 use 2,000 steps; the mod-5 comparisons in this section use 5,000 steps with a cosine schedule spanning that budget. Models use the same width- μ P recipe and four-thread CPU FP32 execution. For RLT-1 and RLT-2, TBPTT128 covers every training sequence, so no gradients are truncated.

Parity trains on 3–40 bits; S_5 trains on 32 operations, using either all 120 permutations or the identity and ten single transpositions (swaps), following [Grazzi et al. \(2024\)](#). Flat mod-5 expressions use $+$, $-$, and $\times$ with multiplication precedence and odd training lengths 3–39; bracketed expressions use trees of lengths 3–40. We score the final answer or state on 1,024 shared test examples per length. Addition trains on randomly sampled 1–8-digit operands and uses teacher-forced answer-token accuracy on 256 shared pairs per test width. For each run, length generalization uses the checkpoint with minimum in-distribution (ID) validation example loss; exact ties select the earliest step. Test scores do not enter checkpoint selection. Means and sample standard deviations (SD, $n = 3$) describe initialization variability on the fixed data stream. Appendices J and J.6 specify generators, metrics, and checkpoint steps.

3.2 Length extrapolation and decoder allocation

RLT-1 sustains parity and swaps- S_5 accuracy well beyond the training lengths (Figure 6). At 256 bits, parity splits $5 + 3$ and $7 + 1$ retain $100 \pm 0\%$ accuracy, compared with $50.07 \pm 1.63\%$ for the

Transformer. The advantage also appears during learning: at step 500, $6 + 2$ reaches $99.44 \pm 0.98\%$ validation accuracy versus $48.48 \pm 0.53\%$ for the Transformer (Figure 7). In a separate seed-42 sixteen-layer series, RLT-1 Layer $8 + 8$, $9 + 7$, $11 + 5$, and $16 + 0$ reach 100% at 256 bits, compared with 49.41% for GPT Layer 16 (Appendix J.8).

Swaps- S_5 favors a larger decoder allocation. At 256 operations, eight times the training length, $4 + 4$ reaches $97.30 \pm 2.76\%$ final-state accuracy, compared with $0.85 \pm 0.30\%$ for the Transformer. At 512 operations, $4 + 4$ still reaches $55.70 \pm 25.78\%$, while the Transformer stays at $0.85 \pm 0.30\%$. Splits $7 + 1$ and $8 + 0$ are already near the uniform reference at 256 operations, despite high training-length accuracy. Figure 13 also reports prefix-token and whole-sequence accuracy.

After 5,000 steps, RLT-1 leads on modular arithmetic at intermediate test lengths. On flat expressions of length 63, RLT-1 Layer $6 + 2$ reaches $93.36 \pm 5.69\%$, compared with $33.20 \pm 2.33\%$ for the Transformer. On bracketed expressions of length 64, $5 + 3$ reaches $67.97 \pm 2.91\%$, compared with $46.71 \pm 1.21\%$. Accuracy falls on longer expressions, and several flat mod-5 splits vary widely across seeds.

3.3 Feedback frequency and parallelism

We compare feedback intervals $B = 1$ (RLT-1), $B = 4$ (RLT-2 chunk4), and $B = \infty$ (RLT-0) at every split, using the data, optimizer, and seeds of the RLT-1 runs (Section 2.7). RLT-0 has 787,968 fewer parameters than RLT-1 at each split; RLT-2 has the same parameters. Figure 8 compares all variants at $4 + 4$, and Appendix J.9 gives every split.

At 64 bits, RLT-1 reaches $100 \pm 0\%$ parity accuracy and chunk4 reaches $98.99 \pm 1.66\%$, while RLT-0 remains near chance at $50.23 \pm 3.80\%$. Swaps- S_5 is more sensitive to the feedback interval: at 64 operations, RLT-1 reaches $100 \pm 0\%$ and chunk4 $19.60 \pm 6.54\%$. RLT-0 and the Transformer are near the $1/120$ uniform reference at this length. Relative to RLT-1, four-token chunks lose about one percentage point on 64-bit parity and about 80 points on 64-operation swaps- S_5 .

The effect of chunking also depends on the split (Figures 21 and 22). Chunk4 $4 + 4$ retains $69.34 \pm 24.17\%$ parity accuracy at 256 bits, whereas chunk4 $7 + 1$ and $8 + 0$ are already near chance at 64 bits. On swaps- S_5 , chunk4 $5 + 3$ reaches $79.92 \pm 10.89\%$ at 48 operations and $50.16 \pm 18.77\%$ at 64, compared with $19.60 \pm 6.54\%$ at 64 for chunk4 $4 + 4$.

3.4 Addition, standard S_5 , and scope

All models reach 100% teacher-forced accuracy on the training-width addition validation set, but accuracy drops beyond eight digits; the eight-layer model means at 32 digits range from 14.89% to 16.84%. Standard S_5 remains difficult, with final-state accuracy near the $1/120$ reference across feedback variants (Figure 23). Appendix J gives the complete six-task curves, endpoint tables, and individual parity trajectories; Appendix J.11 gives the addition feedback-scale ablation. These experiments measure supervised algorithmic performance; RL performance is not evaluated.

4 Related Work

Hybrid Transformer–RNN models. Chen et al. (2018) combine a Transformer encoder with an LSTM-based RNMT+ decoder for machine translation. RLT-1 also combines parallel encoding with

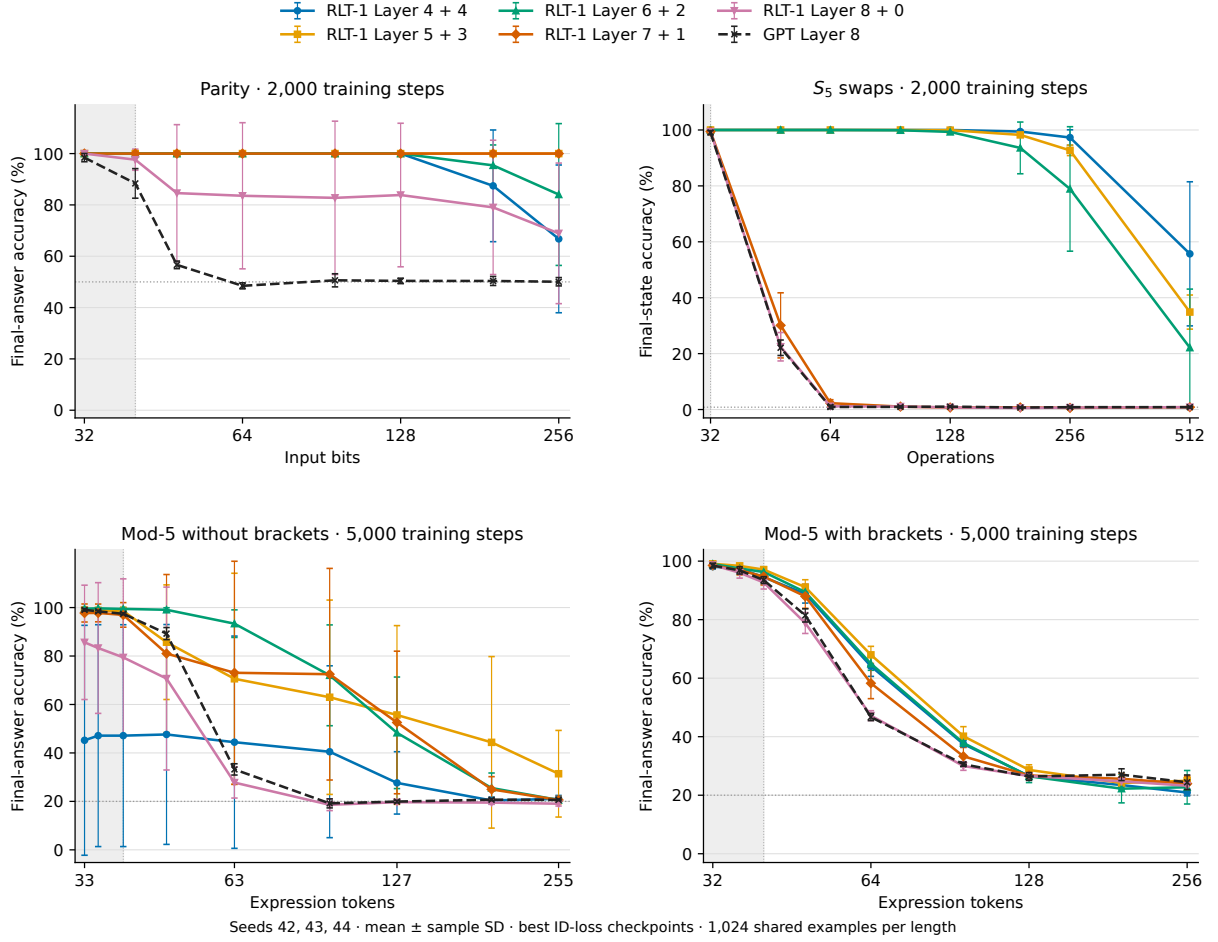


Figure 6 Length generalization across encoder-decoder allocations. All five RLT-1 splits and GPT Layer 8 are shown across the full evaluated length grids. The top row uses 2,000-step parity and swaps- S_5 runs; the bottom row uses 5,000-step mod-5 runs for every model. Each run selects its best ID-loss checkpoint, with earliest-step tie breaking. Points and untrimmed error bars show mean $\pm$ sample SD over seeds 42, 43, and 44 on 1,024 shared examples per length. Gray regions mark trained lengths; dotted horizontal lines mark uniform-prediction accuracy. Flat mod-5 labels actual odd expression lengths. The complete six-task 2,000-step comparison is in Figure 12.

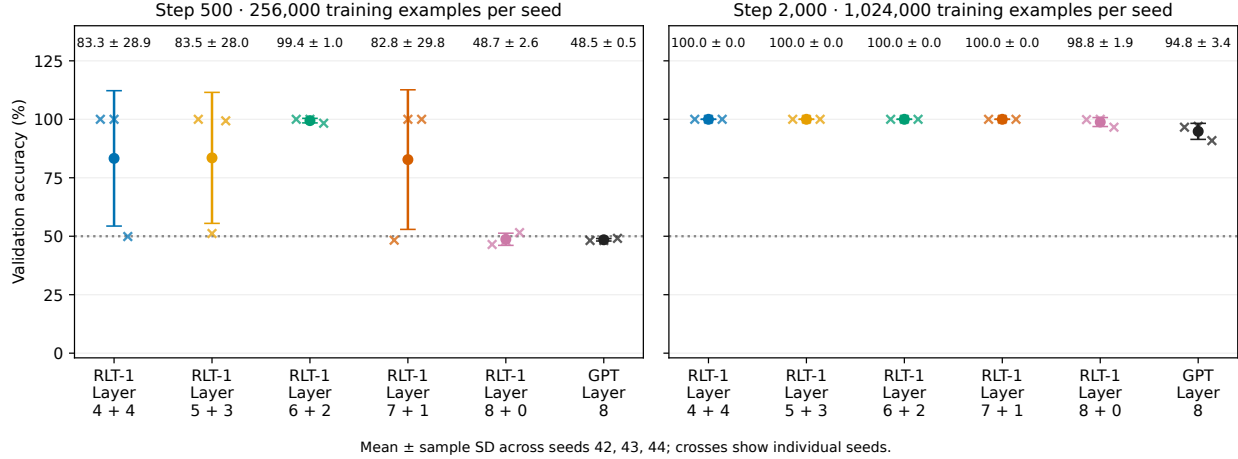


Figure 7 Parity at 500 and 2,000 training steps. Circles and whiskers show the mean and sample SD across seeds 42, 43, and 44; crosses show individual seeds with a small horizontal offset. SD whiskers can extend beyond 100%. Every model uses the same 768 validation examples; the panels correspond to 256,000 and 1,024,000 training examples per seed.

recurrent decoding. Its decoder consists of Transformer blocks with encoder-memory cross-attention and SWA, and it processes both prompt and response tokens in a causal language model.

Encoder-derived memory. YOCO builds reusable KV memory for an upper cross-decoder and allows early exit during prefill (Sun et al., 2024). DeepSeek-V4.1-Flash projects global decoder KV from final encoder states and maintains separate decoder SWA (DeepSeek-AI, 2026). RLT-1 also builds global memory from encoder states, but its recurrent decoder must process every prompt token.

Temporal feedback. Feedback Transformer forms a learned weighted sum of each token’s representations across layers and lets subsequent tokens attend to this shared memory at every layer (Fan et al., 2020). RLT-1 feeds the preceding final decoder output into a gated merge at the decoder input, with separate encoder-derived global memory and layerwise decoder SWA caches. Recurrent Transformer constructs each layer’s persistent KV from that layer’s output and provides an exact tiling schedule to improve memory movement (Oncescu et al., 2026). RLT-1’s recurrent dependency spans the full decoder, whereas Recurrent Transformer’s feedback is layerwise.

Cross-token latent feedback. Full-bandwidth Transformer combines the previous top-layer hidden state with the next token embedding through a gated linear unit, retaining the Transformer stack and KV cache (Wang et al., 2026). Its multi-pass training shifts hidden states between passes to allow token-parallel teacher forcing, with a prefix mix in to address differences between prompts and generation. RLT-1 places output-to-input feedback at the encoder–decoder interface and trains by replaying the decoder sequentially over the full history.

T²MLR feeds a cached middle-layer representation from the previous token into an earlier layer at the current position (Cai et al., 2026). Its experiments find that recurrence between middle layers can outperform recurrence through the full network. Training approximates temporal states with a fixed number of Jacobi iterations and controls backward depth separately. RLT-1 feeds back through the entire decoder and trains by replaying the full history with BPTT, optionally truncated.

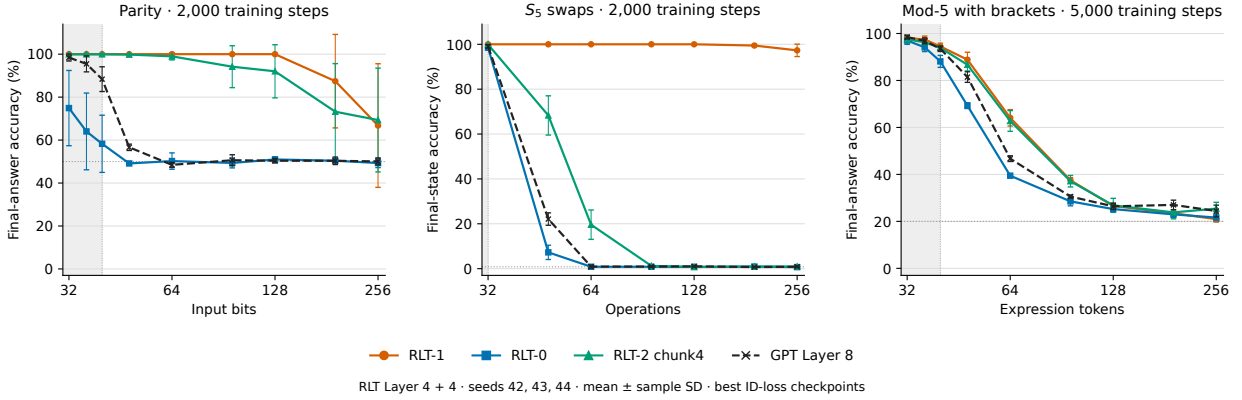


Figure 8 Feedback frequency at a fixed $4 + 4$ split. Final-answer/state accuracy for RLT-1 ($B = 1$), RLT-2 chunk4 ($B = 4$), RLT-0 ($B = \infty$), and GPT Layer 8, across all native test lengths. Parity and swaps- S_5 use 2,000 training steps; bracketed mod-5 uses 5,000. All models use seeds 42–44 and their best ID-loss checkpoints. Error bars show sample SD without clipping; gray regions mark trained lengths and dotted lines mark uniform prediction.

Latent recurrent language models. Latent Recurrent Transformer (LRT) reuses a high-level state from the previous token through KV projection and residual injection, keeping a decoder-only backbone and one forward pass per generated token (Huang et al., 2026). Its interleaved parallel training refines subsets of positions from a shared state buffer. For RL, LRT initializes this buffer with detached rollout states to reduce differences between rollout and recomputation. RLT-1 separates encoder memory from the recurrent decoder and rebuilds the full history under current parameters for exact policy replay.

Continuous latent computation. Coconut feeds the last hidden state back as the next input embedding during a latent reasoning phase, using a curriculum that replaces textual reasoning steps with continuous states (Hao et al., 2024). PonderLM-2 inserts latent steps between ordinary tokens during pretraining and uses Jacobi iterations to approximate their recurrent dependencies in parallel (Zeng et al., 2025). RLT-1 advances its state once per ordinary prompt or response token, without adding latent positions.

Block- and segment-level recurrence. Block-Recurrent Transformers update persistent state vectors with attention and gates, processing a block of tokens in parallel at each recurrent step (Hutchins et al., 2022). Their block-feedback variant lets all layers cross-attend to the recurrent state from the preceding block. Recurrent Memory Transformer appends write-memory tokens to each segment and passes their final representations to the next segment as memory inputs, with BPTT through this connection (Bulatov et al., 2022). RLT-2 is the closest variant: it updates feedback at chunk boundaries from the last token’s final decoder output and gates this state into every decoder input of the next chunk, without dedicated memory tokens or a separate bank of recurrent state vectors. RLT-1, the $B = 1$ case, feeds back at every token and therefore requires sequential decoder work within each block.

Weight sharing across depth. Universal Transformers share weights across depth and optionally adapt the number of refinement steps by position (Dehghani et al., 2018). Saunshi et al. (2025) study how repeated applications of a shared Transformer stack increase effective depth for reasoning,

while recurrent-depth language models vary latent computation at inference time (Geiping et al., 2025). DeepLoop analyzes the effect of repeated parameter visits on residual scaling in weight-tied looped Transformers and derives a loop-aware scaling rule for the Post-LN architecture (Li et al., 2026). These models recur over depth at each position; RLT-1 recurs across tokens, and sharing weights between its encoder and decoder is optional.

5 Conclusion

We introduced the Recurrent Looped Transformer (RLT), which pairs a parallel causal encoder with a decoder that feeds its final state back at every token, so the computation path grows with sequence length at a fixed per-token cost. With eight layers, RLT generalizes parity and swap-based S_5 tracking far beyond the training lengths, where an eight-layer Transformer is at chance, and its best splits outperform the Transformer on modular arithmetic beyond the training lengths. Removing the feedback eliminates the gains on parity and swap-based S_5 . The encoder–decoder split is one design axis: parity generalizes best with one or three decoder layers, whereas swap-based S_5 needs at least two decoder layers and improves with each additional one up to the four tested. The feedback interval B is a second axis that trades parallelism against accuracy: $B = 4$ processes known tokens in parallel within each chunk and keeps 64-bit parity at 99%, while swap-based S_5 benefits most from per-token feedback. Because the chunk size does not change the parameters, a model can be pretrained with large chunks for parallelism and continue with smaller chunks, down to $B = 1$, in mid- and post-training.

Acknowledgement

We used large language models to improve the wording of this work.

References

- Aydar Bulatov, Yuri Kuratov, and Mikhail S. Burtsev. Recurrent memory transformer. *arXiv preprint arXiv:2207.06881*, 2022. URL <https://arxiv.org/abs/2207.06881>.
- Ziyang Cai, Xingyu Zhu, Yihe Dong, Yinghui He, and Sanjeev Arora. T²MLR: Transformer with temporal middle-layer recurrence. *arXiv preprint arXiv:2607.15178*, 2026. URL <https://arxiv.org/abs/2607.15178>.
- Mia Xu Chen, Orhan Firat, Ankur Bapna, Melvin Johnson, Wolfgang Macherey, George Foster, Llion Jones, Niki Parmar, Mike Schuster, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. The best of both worlds: Combining recent advances in neural machine translation. *arXiv preprint arXiv:1804.09849*, 2018. URL <https://arxiv.org/abs/1804.09849>.
- DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. Technical report, DeepSeek-AI, 2026. URL https://huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash/resolve/main/DeepSeek_V41_Tech_Report.pdf. Sections 2.2 and 3.2.2; publicly available from the official DeepSeek model repository.
- Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018. URL <https://arxiv.org/abs/1807.03819>.

- 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. URL <https://arxiv.org/abs/2002.09402>.
- 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. URL <https://arxiv.org/abs/2502.05171>.
- Riccardo Grazi, Julien Siems, Arber Zela, Jörg K. H. Franke, Frank Hutter, and Massimiliano Pontil. Unlocking state-tracking in linear rnns through negative eigenvalues. *arXiv preprint arXiv:2411.12537*, 2024. URL <https://arxiv.org/abs/2411.12537>.
- Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023. URL <https://arxiv.org/abs/2312.00752>.
- Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021. URL <https://arxiv.org/abs/2111.00396>.
- 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. URL <https://arxiv.org/abs/2412.06769>.
- 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>.
- DeLesley Hutchins, Imanol Schlag, Yuhuai Wu, Ethan Dyer, and Behnam Neyshabur. Block-recurrent transformers. *arXiv preprint arXiv:2203.07852*, 2022. URL <https://arxiv.org/abs/2203.07852>.
- Shuzhen Li, Yifan Zhang, Jiacheng Guo, Quanquan Gu, and Mengdi Wang. DeepLoop: Depth scaling for looped transformers. *arXiv preprint arXiv:2607.13491*, 2026. URL <https://arxiv.org/abs/2607.13491>.
- William Merrill, Jackson Petty, and Ashish Sabharwal. The illusion of state in state-space models. *Proceedings of the 41st International Conference on Machine Learning*, 2024. URL <https://arxiv.org/abs/2404.08819>.
- 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. URL <https://arxiv.org/abs/2604.21215>.
- Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, 2025. URL <https://arxiv.org/abs/2502.17416>.
- Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. *arXiv preprint arXiv:2405.05254*, 2024. URL <https://arxiv.org/abs/2405.05254>.
- 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*, 2026. URL <https://arxiv.org/abs/2608.08888>.
- Boyi Zeng, He Li, Shixiang Song, Yixuan Wang, Ziwei He, Xinbing Wang, and Zhouhan Lin. PonderLM-2: Pretraining LLM with latent thoughts in continuous space. *arXiv preprint arXiv:2509.23184*,

2025. URL <https://arxiv.org/abs/2509.23184>.
Yifan Zhang et al. Reliable RL scaling requires accounting for Prefill–Decode kernel mismatch. Technical report, Pretraining-RL-Science project, August 2026. URL https://github.com/yifanzhang-pro/Pretraining-RL-Science/blob/master/Prefill_Decode_Kernel_Mismatch.pdf. Dated August 6, 2026; revised August 24, 2026.

Appendix

A	Architecture and Execution Details	19
A.1	Sharing encoder and decoder weights	19
B	Computational Properties	19
B.1	Prompt–response consistency	19
B.2	Prefill work and sequential depth	19
C	Hardware Execution	20
C.1	Batching independent sequences	20
C.2	Memory traffic and parameter reuse	20
C.3	Training memory and numerical agreement	21
D	Training Objectives	21
D.1	Autoregressive pretraining	21
D.2	Supervised fine-tuning	21
D.3	Policy gradients and replay	21
E	Multi-turn Inference and Cache Reuse	22
F	Execution Procedures	24
F.1	Recurrent prompt prefill	24
F.2	Generation and new external inputs	24
F.3	Pretraining and SFT	24
F.4	Current-policy RL replay	25
G	Causality and Gradient Paths	25
H	When Cached States Can Be Reused	26
I	RLT-2: Chunk-Parallel Hidden-State Feedback	27
I.1	Chunk operator and causal masks	27
I.2	Training and prefill algorithm	27
I.3	Incremental generation and partial chunks	28
I.4	Training and inference efficiency	29
J	Experimental Details	29
J.1	Complete six-task study at 2,000 steps	29
J.2	Parameter counts	36
J.3	Implementation of RLT-0	36
J.4	Seed aggregation and metric definitions	36
J.5	Reproducing the completed training snapshot	36
J.6	Length-generalization protocol and supplementary metrics	39
J.7	Token-accuracy trajectories at lengths 32 and 33	43
J.8	Sixteen-layer parity: best and final checkpoints	44
J.9	Three-seed comparison of feedback variants	46
J.10	Mod-5 feedback variants at 5,000 steps	51
J.11	Addition feedback scale at a fixed learning-rate schedule	56

A Architecture and Execution Details

A.1 Sharing encoder and decoder weights

The reference tied RLT-1 sets $L_E = L_D = L$. Encoder self-attention at layer ℓ and decoder SWA at layer ℓ share compatible query, key, value, and output projections; the corresponding FFNs also share weights. The encoder attends to its causal context; decoder SWA attends to decoder activations within its window. Decoder cross-attention uses separate query/output projections and the memory projections of Equation (2.2), adding computation beyond the shared attention and FFN. The encoder and decoder retain separate normalizations, and the merge and readout are separate modules.

The two passes share weights but compute separate activations. An untied E_θ, D_ϕ uses separate weights and keeps the same recurrence. Memory groups can be shared in either version; each decoder layer still maintains its own SWA cache.

B Computational Properties

B.1 Prompt–response consistency

Proposition B.1 (Invariance to the serving split). Fix the parameters, token sequence, position convention, and initial state for an independent sequence. Assume exact arithmetic, mathematically equivalent causal encoder execution, identical SWA windows and cache updates, and deterministic decoder operations. Processing a prefix with batched encoder prefill followed by recurrent decoder updates gives the same states and next-token distributions as processing it incrementally. The conditional distribution for a fixed token history is therefore unchanged when the prompt–response split moves.

Proof. Causal encoder equivalence gives the same e_t and $M_{\leq t}$ in both schedules. Both start from $H_0 = (s_*, \emptyset)$. If their states agree at $t - 1$, Equation (2.5) applies the same operations to the same inputs at t , so their states agree at t . The result follows by induction, since the transition does not depend on the serving split. $\square$

B.2 Prefill work and sequential depth

Let $C_E^{\text{pf}}(T)$ be encoder prefill work, $C_M(T)$ the memory projection work, and $C_D^{\text{step}}(t)$ a decoder evaluation over t encoder-memory entries and at most W decoder positions per layer, including merge overhead. Then

$$C_{\text{prefill}}(T) = C_E^{\text{pf}}(T) + C_M(T) + \sum_{t=1}^T C_D^{\text{step}}(t). \quad (\text{B.1})$$

For dense attention and width-proportional KV, a coarse arithmetic estimate is

$$O((L_E + L_D)(Td^2 + T^2d) + GTd^2 + L_DT \min(W, T)d). \quad (\text{B.2})$$

Prefill requires T sequential decoder transitions, each containing L_D blocks. These sequential dependencies can reduce hardware utilization even when arithmetic complexity is of the same order as a dense Transformer’s.

Each new token evaluates $L_E + L_D$ blocks, plus merge and memory projection. Attention work still grows with context length. With effective KV width d_{KV} , inference cache storage is approximately

$$O((L_E + G)t d_{KV} + L_D \min(t, W - 1)d_{KV}^D + d). \quad (\text{B.3})$$

Here d_{KV}^D is the effective decoder SWA KV width, and the $O(d)$ term stores the current recurrent output. The SWA term counts retained history; current-position KV and training activations need additional storage. Weight sharing reduces parameter storage but retains both passes and their caches.

C Hardware Execution

C.1 Batching independent sequences

For a known training sequence or prompt, the encoder computes features and memory projections in parallel across positions, then the decoder processes positions in order. Independent sequences can batch their next decoder updates together (Figure 9). Each sequence keeps its own encoder prefix, SWA cache, recurrent output, and position.

At inference, each step consists of an encoder update, memory append, merge, decoder pass, and readout. Batching requests increases matrix-operation sizes and allows weight reuse. Small batches and uneven sequence lengths may limit utilization.

Batching decoder updates from independent sequences

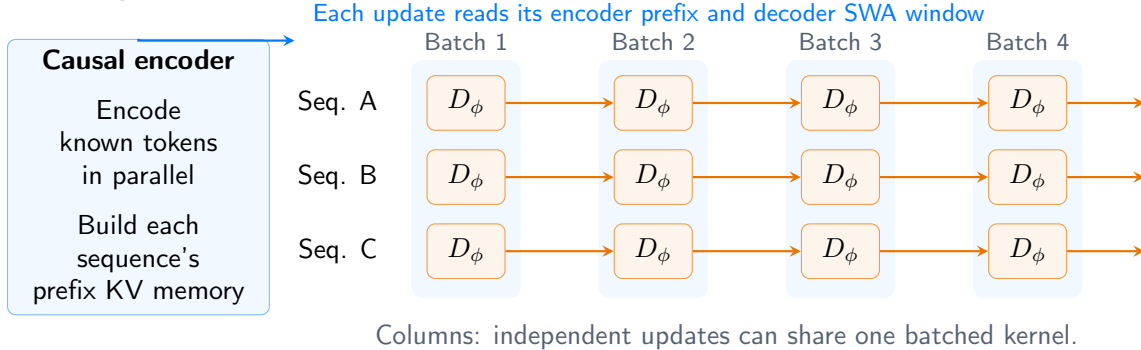


Figure 9 Batching decoder updates across sequences. Columns group independent updates into a batch; rows follow each sequence in token order. Each update reads its own encoder prefix and decoder SWA window. Known tokens can be encoded in parallel before replay; generated tokens are encoded as they arrive.

C.2 Memory traffic and parameter reuse

Encoder KV stays fixed while the prefix and parameters are unchanged. Decoder layers reuse that memory and append their own KV to separate SWA caches. Fewer memory groups reduce KV storage and projection work; more groups allow different transformations for each layer. Sharing encoder and decoder weights may also help keep weights resident on the accelerator.

Normalization, gating, state projection, and residual addition can be fused into one kernel.

C.3 Training memory and numerical agreement

Activation checkpointing reduces stored activations by recomputing them during backpropagation through time (BPTT).

Zhang et al. (2026) describe how the executed policy depends on precision, cache construction, reductions, and sampling transforms as well as weights. Sampler and trainer implementations must align positional conventions, stochastic behavior, and cache contents, and check numerical agreement across execution modes.

D Training Objectives

D.1 Autoregressive pretraining

Full-sequence next-token prediction is the base objective:

$$\mathcal{L}_{\text{PT}}(\Theta) = -\mathbb{E}_{x_{1:S}} \left[\frac{1}{S-1} \sum_{t=1}^{S-1} \log p_{\Theta}(x_{t+1} \mid x_{1:t}) \right]. \quad (\text{D.1})$$

Initialize $H_0 = (s_*, \emptyset)$, compute causal encoder features, and update the decoder at positions $1, \dots, S-1$. Every non-BOS target contributes to the loss; the encoder, memory projections, merge, and decoder train jointly. At each independent document, reset both decoder and encoder state, reset positions, and prevent attention across document boundaries. When training on segments, specify the initial context and state: dropping an earlier state changes the history on which the likelihood is conditioned.

D.2 Supervised fine-tuning

Let $m_{t+1} = 1$ for assistant targets and 0 for user, system, tool, or padding targets. For examples with at least one selected target, optimize

$$\mathcal{L}_{\text{SFT}}(\Theta) = -\mathbb{E}_x \left[\frac{1}{\sum_{t=1}^{S-1} m_{t+1}} \sum_{\substack{1 \leq t < S \\ m_{t+1}=1}} \log p_{\Theta}(x_{t+1} \mid x_{1:t}) \right]. \quad (\text{D.2})$$

The mask selects loss terms; user, system, and tool tokens still update the decoder state. Gradients from assistant losses flow through these updates, including encoder memory and decoder KV. The model continues from the existing state when an assistant turn begins.

D.3 Policy gradients and replay

Let $c = x_{1:T}$ be a prompt and $y = (y_1, \dots, y_N)$ a sampled response, with $y_i = x_{T+i}$. The policy is

$$\pi_{\Theta}(y \mid c) = \prod_{i=1}^N p_{\Theta}(y_i \mid c, y_{<i}). \quad (\text{D.3})$$

For a sequence reward $R(c, y)$ independent of Θ , define $J(\Theta) = \mathbb{E}_{c, y \sim \pi_\Theta} [R(c, y)]$. Its on-policy score-function gradient is

$$\nabla_\Theta J = \mathbb{E}_{c, y \sim \pi_\Theta} \left[(R(c, y) - b(c)) \sum_{i=1}^N \nabla_\Theta \log p_\Theta(y_i \mid c, y_{<i}) \right], \quad (\text{D.4})$$

where $b(c)$ is a response-independent baseline, held constant when taking the policy gradient. During replay, the sampled tokens are fixed and gradients pass through the recurrent computation used to evaluate their log-probabilities.

Figure 10 shows the replay procedure. The sampler records each action’s log-probability under the distribution μ that sampled it. The trainer rebuilds encoder features, recurrent outputs, and decoder SWA caches under the current parameters Θ , starting from the initial state and processing the full prompt and sampled response prefix. The resulting action probabilities give the ratios

$$r_i(\Theta) = \exp(\log p_\Theta(y_i \mid c, y_{<i}) - \log \mu(y_i \mid c, y_{<i})). \quad (\text{D.5})$$

Here the target is the raw model policy p_Θ . The behavior probabilities must include any temperature scaling, truncation, and renormalization used by the sampler. Exact importance sampling requires $p_\Theta(\cdot \mid h) \ll \mu(\cdot \mid h)$: at each relevant history, every action with positive target probability must also have positive behavior probability. Top- k or top- p sampling generally excludes actions with positive probability under an untruncated softmax target. If the target policy itself uses a sampling transform, apply it consistently in the objective and ratios.

To preserve both forward probabilities and their parameter derivatives, the trainer differentiates the recurrent computation or an equivalent implementation (Zhang et al., 2026). The recorded behavior log-probabilities stay fixed across parameter updates, since they describe the policy that sampled the actions.

For fixed prompts and sequence rewards, exact trajectory importance sampling uses $\prod_i r_i$, subject to support and integrability conditions. Tokenwise clipping and other surrogate losses may introduce bias even with correct replay probabilities.

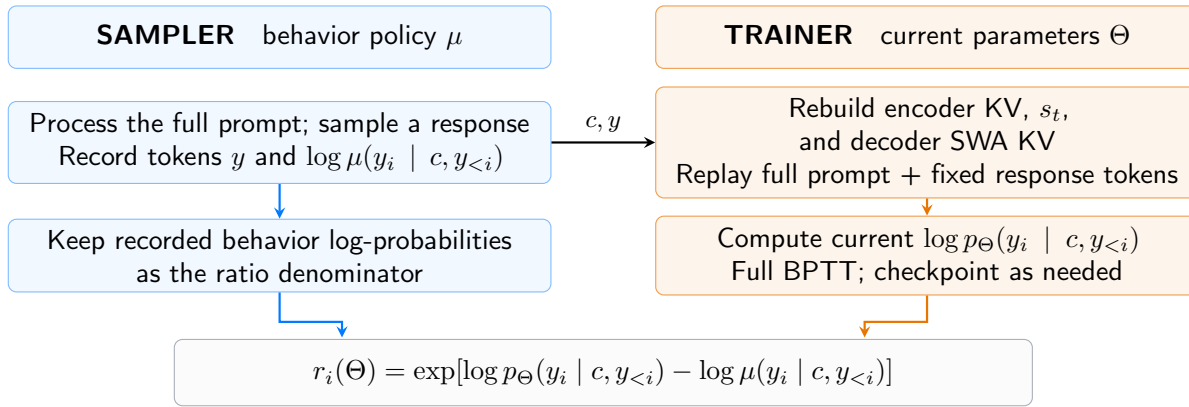
E Multi-turn Inference and Cache Reuse

A conversation forms one token history beginning at BOS. User messages, tool results, role delimiters, and assistant tokens all receive encoder and decoder updates. For example, when a tool returns, encode its output using the cached prefix, then process those tokens through the decoder in order before resuming generation. Reset state only for a new independent sequence or an explicitly defined context reset.

To resume from a saved prefix, retain the encoder cache C_t^E , cross-attention memory $M_{\leq t}$, decoder state $H_t = (s_t, C_t^D)$, tokens and positions, SWA window convention, and model version. These values can be reused at fixed weights regardless of where the prompt ends. Reproducing the same sampled outputs also requires the sampler’s random state. If only text is saved, replay it to rebuild the state.

Editing or truncating an earlier token invalidates the later cached state. Recompute from a valid checkpoint before the edit; this also applies to chat templates that rewrite earlier tokens. In multi-turn RL, user and tool tokens update the state and carry gradients, but receive no action importance-ratio factors because the policy did not sample them.

Replaying a sampled history under current parameters



After every parameter update: rebuild current-policy KV and states; retain original behavior log-probabilities.

Figure 10 Replaying a sampled history under current parameters. The sampler records behavior probabilities. The trainer rebuilds the full prefix, including recurrent outputs and decoder SWA caches, to evaluate the current policy. The ratio compares the current action probability with the recorded behavior probability.

F Execution Procedures

The complete decoder state is $H_t = (s_t, C_t^D)$, initialized as $H_0 = (s_*, \emptyset)$. Encoder continuation state and encoder-derived cross-attention memory are maintained separately. All schedules use the block order and window convention in Equation (2.20).

F.1 Recurrent prompt prefill

1. Start an independent sequence with $x_1 = \text{BOS}$ and $H_0 = (s_*, \emptyset)$; initialize encoder caches and positions.
2. Encode $x_{1:T}$ causally in parallel and construct its encoder KV memory.
3. For $t = 1, \dots, T$, compute $H_t = D_\phi(\text{Merge}(e_t, s_{t-1}); M_{\leq t}, C_{t-1}^D, t)$. Each decoder layer reads current KV and the retained SWA history. Both attention masks exclude positions after t .
4. Predict the first response token from s_T . Preserve H_T , encoder caches, memory, and positional metadata for the next update.

F.2 Generation and new external inputs

Sample x_{t+1} from the distribution predicted by s_t , then update the encoder cache, memory, and complete decoder state to H_{t+1} . This state predicts x_{t+2} . Each consumed token receives exactly one recurrent update and one KV insertion at every decoder SWA layer.

Multiple tokens from a user or tool can be encoded in a causal batch conditioned on the existing prefix. The decoder processes them in token order from the saved state, updating both s_t and C_t^D .

If generation stops at a length limit before consuming its last emitted token, record that token separately. The cache represents the consumed prefix; process the pending token once before continuing with later tokens.

F.3 Pretraining and SFT

1. Compute $e_{1:S-1}$ using a causal encoder and construct memory.
2. Initialize $H_0 = (s_*, \emptyset)$; unroll all positions $1, \dots, S - 1$, including every layerwise SWA cache update.
3. Accumulate all valid next-token losses for pretraining, or only assistant-target losses for SFT. All context tokens receive state updates.
4. Normalize each example by its number of selected targets, then average examples as in Equations (D.1) and (D.2). Backpropagate through the complete computation, using checkpointing if required.

Examples without selected targets are excluded from the loss average. Token averaging would give more weight to examples with more selected targets.

Packed independent sequences need separate recurrent states, SWA caches, encoder caches, positions, and attention masks. Both attention mechanisms must stay within document boundaries, even where losses are masked.

F.4 Current-policy RL replay

1. Read the rollout token history, action mask, actual behavior log-probabilities, and sampling metadata. Behavior probabilities include all sampling transforms and renormalization.
2. Hold current parameter values fixed throughout forward replay and backward. Use the target policy’s positional, SWA, and execution conventions, with parameter gradients enabled.
3. Recompute encoder representations of the known history. From $H_0 = (s_*, \emptyset)$, rebuild the recurrent output and every decoder SWA cache through all prompt tokens.
4. Replay subsequent tokens in order. Before consuming each sampled action, read its current-policy log-probability from the preceding state. Include EOS if sampled. Consume every intervening external token needed for later predictions.
5. Form the chosen RL loss from action log-probabilities, rewards or advantages, and any required behavior ratios. Apply action factors only to tokens sampled by the policy.
6. Backpropagate, update parameters, and invalidate parameter-dependent encoder and decoder caches before the next exact current-policy replay.

The action mask selects loss terms while preserving gradients through user and tool tokens. Prompt replay under `no_grad`, or detaching prompt KV, preserves forward values but drops the prompt-state gradients required by full BPTT.

A deterministic forward pass, for example with dropout disabled, gives a reproducible probability for each action. For a stochastic policy, specify whether action probabilities condition on internal randomness or average over it. Each replay evaluates one realization; marginal probabilities require averaging over that randomness.

G Causality and Gradient Paths

Proposition G.1 (Causality). With a causal encoder, prefix-restricted encoder memory, and causal decoder SWA, $H_t = (s_t, C_t^D)$ depends only on $x_{1:t}$ and the parameters, including s_* , under deterministic execution.

Proof. Encoder causality implies that each e_j depends only on $x_{1:j}$, hence $M_{\leq t}$ depends only on $x_{1:t}$. The initial state contains no future-token information. Suppose H_{t-1} depends only on $x_{1:t-1}$. The next update uses this state, e_t , and $M_{\leq t}$. At each decoder layer, current KV is formed from the causally available layer input, and SWA reads no position greater than t . Appending current KV and evicting old entries introduce no future information. Thus both s_t and C_t^D depend only on $x_{1:t}$, completing the induction. $\square$

For teacher-forced encoder features held fixed, use a fixed-slot representation of the decoder cache, with validity masks during warm-up, and define

$$\mathcal{J}_t = \frac{\partial H_t}{\partial H_{t-1}}. \quad (\text{G.1})$$

Then

$$\frac{\partial H_t}{\partial H_j} = \mathcal{J}_t \mathcal{J}_{t-1} \cdots \mathcal{J}_{j+1}, \quad j < t, \quad (\text{G.2})$$

where

$$\mathcal{J}_t = \begin{pmatrix} \frac{\partial s_t}{\partial s_{t-1}} & \frac{\partial s_t}{\partial C_{t-1}^D} \\ \frac{\partial C_t^D}{\partial s_{t-1}} & \frac{\partial C_t^D}{\partial C_{t-1}^D} \end{pmatrix}. \quad (\text{G.3})$$

The block Jacobian includes paths through both the recurrent output and decoder KV.

Encoder memory retains information from past encoder outputs; decoder SWA provides direct access to recent decoder projections. Evicted decoder entries may still affect later computation through states or activations that read them before eviction.

Full parameter derivatives also include the parameter dependence of encoder features, memory, every transition, and decoder KV projections. Let B_T^E denote all encoder-side boundary tensors needed for response continuation, including encoder continuation KV and encoder-derived cross-attention memory. For a response loss $\ell(\Theta, H_T(\Theta), B_T^E(\Theta))$, the chain rule gives

$$\frac{d\ell}{d\Theta} = \frac{\partial \ell}{\partial \Theta} + \frac{\partial \ell}{\partial H_T} \frac{\partial H_T}{\partial \Theta} + \frac{\partial \ell}{\partial B_T^E} \frac{\partial B_T^E}{\partial \Theta}. \quad (\text{G.4})$$

The first term holds the boundary arguments fixed; the others account for their prefix computation. In particular,

$$\frac{\partial \ell}{\partial H_T} \frac{\partial H_T}{\partial \Theta} = \frac{\partial \ell}{\partial s_T} \frac{\partial s_T}{\partial \Theta} + \frac{\partial \ell}{\partial C_T^D} \frac{\partial C_T^D}{\partial \Theta}. \quad (\text{G.5})$$

Detaching s_T , decoder KV, or encoder boundary tensors removes the corresponding gradient terms, even when forward probabilities stay unchanged.

H When Cached States Can Be Reused

A saved prefix can replace forward replay when it matches the current parameters, consumed tokens, positions, initialization, SWA window, and policy execution settings. Save the recurrent output, each decoder SWA cache, encoder continuation state, encoder memory, and associated metadata. The saved prefix may come from the sequence beginning or from an exact checkpoint made with the current parameters and execution settings.

A detached cache can hold correct values while lacking the computation graph needed for training. Full BPTT requires retaining or recomputing the prefix graph so gradients reach every parameter-dependent boundary tensor.

Activation checkpointing within one update recomputes activations under the same parameters and stochastic state. Mutable cache implementations must restore the saved contents during recomputation and preserve activations needed for backward.

Truncated BPTT preserves numeric values while cutting selected gradient dependencies. A complete decoder-state detach is

$$\tilde{H}_t = (\text{stopgrad}(s_t), \text{stopgrad}(C_t^D)). \quad (\text{H.1})$$

Detaching only s_t leaves possible paths through decoder KV; detaching only decoder KV leaves paths through the recurrent output. Encoder-side boundary tensors can provide additional paths across the same boundary.

SWA eviction removes old KV from future attention windows. Full BPTT still differentiates earlier computations that used those entries, so training may need more activation storage than the inference cache alone.

After a weight update, recompute the prefix values and gradient graph before the next exact training pass.

I RLT-2: Chunk-Parallel Hidden-State Feedback

Section 2.7 defines the RLT-2 merge and boundary update, and Figure 4 shows one chunk; this appendix gives its masks, training and prefill procedure, incremental generation, and costs. Section 3 and Appendix J.9 report the experiments.

I.1 Chunk operator and causal masks

Choose a fixed chunk size $B \geq 1$, count BOS as position one, and anchor boundaries to the start of each independent sequence. For a known prefix of length T , define

$$K = \lceil T/B \rceil, \quad a_k = (k-1)B + 1, \quad b_k = \min(kB, T), \quad I_k = \{a_k, \dots, b_k\}. \quad (\text{I.1})$$

Starting from empty decoder caches, the merged inputs $z_{I_k}^0$ of Equation (2.22) pass through the chunk operator and readout:

$$y_{I_k}, C_{b_k}^D = D_\phi^{\text{chunk}}(z_{I_k}^0; M, C_{a_k-1}^D, I_k), \quad (\text{I.2})$$

$$p_\Theta(x_{t+1} \mid x_{1:t}) = \text{softmax}(W_o \text{RMSNorm}_o(y_t))_{x_{t+1}}, \quad t \in I_k. \quad (\text{I.3})$$

The chunk operator applies Equation (2.20) layer by layer, using all known positions in I_k together. At each layer, query t reads decoder keys at $\max(1, t - W + 1) \leq j \leq t$, including retained keys from earlier chunks, and encoder memory only at $j \leq t$. Neither SWA nor encoder positions reset at a chunk boundary. The chunk may be longer or shorter than the SWA window.

Each decoder layer projects KV for all positions from that layer’s inputs before masked attention, so causal SWA can process those positions in parallel. The last-position output carries the chunk’s causal computation to the next chunk, without pooling or an additional summary token. A sequence fitting in one chunk still receives the learned initial-state merge, whereas RLT-0 removes that merge and its parameters.

I.2 Training and prefill algorithm

For teacher-forced training, $x_{1:T}$ denotes the consumed input tokens, with a next-token target wherever one is available. The same forward procedure computes prompt prefill and known-history policy replay:

1. Encode the known tokens causally and build encoder KV memory. Initialize $h = s_\star$, empty decoder SWA caches, and global position one.
2. Visit chunks in increasing order. Compute $\text{RMSNorm}_s(h)$ and $W_s \text{RMSNorm}_s(h)$ once for the chunk; broadcast them to its token-specific gated merges.

3. For each decoder layer, project the chunk’s Q/K/V together, apply causal SWA using retained and current-chunk KV, then apply prefix-masked memory attention and the FFN. Retain the last $W - 1$ decoder KV entries per layer for continuation.
4. Compute logits and the selected next-token losses at every position. After a complete chunk, set h to its last decoder output. A partial final chunk leaves h at the preceding complete boundary.
5. For training, accumulate the sequence loss and backpropagate through all chunks before updating parameters. For prefill, save caches, h , the consumed-token count, and the last output or next-token logits.

Full BPTT differentiates through the boundary state, cross-chunk decoder KV, and encoder memory. Chunking changes the forward dependencies; gradient truncation and optimizer-update frequency are separate choices. For RL, replay uses the same B , boundary anchor, masks, and positions as sampling, rebuilding the state under the current parameters before evaluating action probabilities.

I.3 Incremental generation and partial chunks

Let t be the number of consumed tokens and $r = t \bmod B$ the current chunk offset. The saved feedback register is $h_{\lfloor t/B \rfloor}$, including when the prompt ends inside a chunk. To consume the next observed or sampled token at position $j = t + 1$:

1. Update the causal encoder and append its memory KV. Merge e_j with the saved boundary state, keeping that state fixed throughout the chunk.
2. Run one decoder step with the usual causal SWA caches and prefix memory, producing y_j , updated KV caches, and next-token logits.
3. If $j \bmod B = 0$, replace the feedback register by y_j ; otherwise preserve it. Advance the consumed-token count to j .

The latest y_j always predicts the next token, whether or not it updates the feedback register. For example, with $B = 4$ and a six-token prompt, tokens 5–8 all merge with $h_1 = y_4$; processing tokens 5 and 6 during prefill does not replace that register with y_6 . The output y_6 predicts token 7, and y_8 becomes h_2 only after token 8 is consumed. Known continuation tokens can be batched up to the next fixed boundary, then the procedure continues with the new state. Padding, message boundaries, and the prompt–response split do not commit a partial chunk. Independent packed sequences each maintain their own boundary anchor and state.

With fixed weights and exact arithmetic, batched and incremental execution agree if they use the same mathematical operators and stochastic behavior. Stochastic operations must be disabled or use identical random draws for corresponding operations. Each position then has the same encoder prefix, preceding boundary state, and causally available layerwise KV in both schedules. Induction over layers within a chunk and then over chunks gives the same outputs and boundary states. Changing B or shifting the boundary anchor changes the model computation and invalidates a saved continuation state.

Chunkwise prefill and tokenwise decoding can use different matrix and attention kernels and floating-point reduction orders, altering logits and boundary states even under deterministic execution. For RL, compare chunkwise replay against a tokenwise reference that reproduces the rollout’s encoder, decoder, and readout execution, including precision, batch layout, kernel choices, and stochastic settings.

I.4 Training and inference efficiency

Table 1 assumes equal encoder and decoder depths, width, windows, and memory groups, with $L_D \geq 1$. Its decoder block stages count sequential dependencies in the layerwise schedules, excluding attention-kernel reductions, communication, and hardware-dependent latency.

At fixed dimensions, the common forward arithmetic with dense encoder and memory attention is

$$O((L_E + L_D)Td^2 + (L_E + L_D)T^2d + GTd^2 + L_DT \min(W, T)d). \quad (\text{I.4})$$

RLT-1 adds $O(Td^2)$ merge work. RLT-2 also adds $O(Td^2)$ merge work for T token-specific gates, but reuses state normalization and projection within each chunk. Its K sequential decoder batches permit larger matrix operations and weight reuse across up to B positions. Total decoder arithmetic still covers all T tokens, and full BPTT requires activations or recomputation across all chunks.

During generation at context length t , each variant evaluates its encoder and decoder stacks per token and reads the available encoder memory. RLT-2 can reuse its feedback projection until the next boundary; the token-specific gate, decoder blocks, and KV writes remain per-token operations. The shared inference-cache scaling is

$$O((L_E + G)t d_{KV} + L_D \min(t, W - 1)d_{KV}^D). \quad (\text{I.5})$$

RLT-1 and RLT-2 add $O(d)$ feedback storage; RLT-2 also retains a chunk offset and may cache $O(d)$ normalized/projected state. Chunk batches require temporary activations and current-chunk KV during prefill.

A controlled benchmark would sweep $B \in \{1, 2, 4, 8, 16, 32\}$ and RLT-0 at fixed data, stack dimensions, precision, optimizer, batch size, and attention kernels. It would measure task accuracy, training throughput, prefill latency by prompt length, generation time per token by cached context length, and peak device memory.

The $B = 1$ run would check RLT-1 equivalence, and future-token perturbations would check causality. At unchanged parameters, maximum and RMS differences in logits, selected-action log-probabilities, and boundary states across chunkwise, partial-chunk, and tokenwise execution would separate execution differences from the effects of a policy update. A differentiable reference should also compare full-BPTT parameter gradients, including paths through boundary states, decoder KV, and encoder memory.

J Experimental Details

J.1 Complete six-task study at 2,000 steps

This appendix reports all 108 runs of the eight-layer study: six architectures on six tasks with initialization seeds 42, 43, and 44, each trained for 2,000 optimizer steps. It gives validation curves at training lengths and length generalization from each run’s best in-distribution checkpoint.

J.1.1 Models, tasks, and evaluation protocol

Models, optimizer settings, batch sizes, and execution follow Section 3; the cosine schedule reaches 5×10^{-6} at step 2,000. Validation runs every 100 steps, and each run sees 1,024,000 training examples.

Table 2 Validation accuracy (%) after 2,000 optimizer steps. All entries are mean $\pm$ sample SD over seeds 42, 43, and 44. Addition uses teacher-forced answer tokens; the formal tasks use final labels or states. Every run has consumed 1,024,000 training examples.

Task	RLT-1 Layer 4 + 4	RLT-1 Layer 5 + 3	RLT-1 Layer 6 + 2	RLT-1 Layer 7 + 1	RLT-1 Layer 8 + 0	GPT Layer 8
Addition	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00
Parity	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	98.83 $\pm$ 1.92	94.84 $\pm$ 3.43
Mod 5, no brackets	45.36 $\pm$ 46.43	94.18 $\pm$ 7.29	69.62 $\pm$ 33.01	70.01 $\pm$ 43.15	60.33 $\pm$ 35.70	64.02 $\pm$ 37.64
Mod 5, brackets	70.53 $\pm$ 10.75	74.35 $\pm$ 6.71	74.87 $\pm$ 3.72	74.78 $\pm$ 3.01	75.17 $\pm$ 6.78	73.87 $\pm$ 9.07
S_5 , swaps	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	99.35 $\pm$ 0.81	99.61 $\pm$ 0.68	99.09 $\pm$ 0.23
S_5 , standard	0.78 $\pm$ 0.68	2.47 $\pm$ 1.26	2.21 $\pm$ 1.48	2.08 $\pm$ 0.98	1.82 $\pm$ 1.19	0.52 $\pm$ 0.23

- **Addition:** randomly sample 1–8-digit operands and serialize their digits in reverse order. The 256 validation examples use the same range. We measure teacher-forced answer-token accuracy: each prediction receives the correct preceding tokens, and scoring includes answer formatting and EOS while excluding prompt and padding positions.
- **Parity:** train on binary strings of lengths 3–40 and supervise the final parity bit. Validation pools 256 examples at each of lengths 32, 36, and 40.
- **Modular arithmetic:** evaluate expressions modulo five using $+$, $-$, and $\times$. The flat variant respects multiplication precedence and trains on odd expression lengths 3–39, with validation lengths 33, 35, and 39. The bracketed variant uses expression trees, trains on lengths 3–40, and validates at 32, 36, and 40. Each validation length has 256 examples; only the final value is supervised.
- **S_5 state tracking:** compose 32 permutations and supervise the running product after each input, following [Grazzi et al. \(2024\)](#). Standard inputs range over all 120 permutations; the swaps variant uses the identity and the ten single transpositions. Each variant has 256 validation sequences of length 32. We report final-state accuracy as the primary metric and prefix-token accuracy in Table 5.

For each run, we pool counts across its configured validation lengths before averaging across initializations. Figure 11 shows all three seeds at each optimizer step, and Table 2 compares every model at step 2,000.

J.1.2 Parity: learning speed and initialization variability

At 500 steps, RLT-1 Layer 6+2 reaches $99.44 \pm 0.98\%$ validation accuracy, compared with $48.48 \pm 0.53\%$ for the Transformer. RLT-1 Layer 4 + 4, 5 + 3, and 7 + 1 average about 83%, with SDs of 28–30 percentage points; 8 + 0 remains near chance. All three 6 + 2 seeds reach 100% by step 600. At step 2,000, splits 4 + 4 through 7 + 1 reach $100 \pm 0\%$, while 8 + 0 reaches $98.83 \pm 1.92\%$ and the Transformer $94.84 \pm 3.43\%$. Figure 7 compares the early and final checkpoints.

J.1.3 Modular arithmetic with and without brackets

Flat modular arithmetic shows substantial initialization variability at 2,000 steps. RLT-1 Layer 5 + 3 has the highest mean, $94.18 \pm 7.29\%$, compared with $64.02 \pm 37.64\%$ for the Transformer. The 4 + 4 scores are 17.58%, 19.53%, and 98.96% for seeds 42, 43, and 44; their mean is $45.36 \pm 46.43\%$. At this training budget, several architectures produce both successful and near-chance runs.

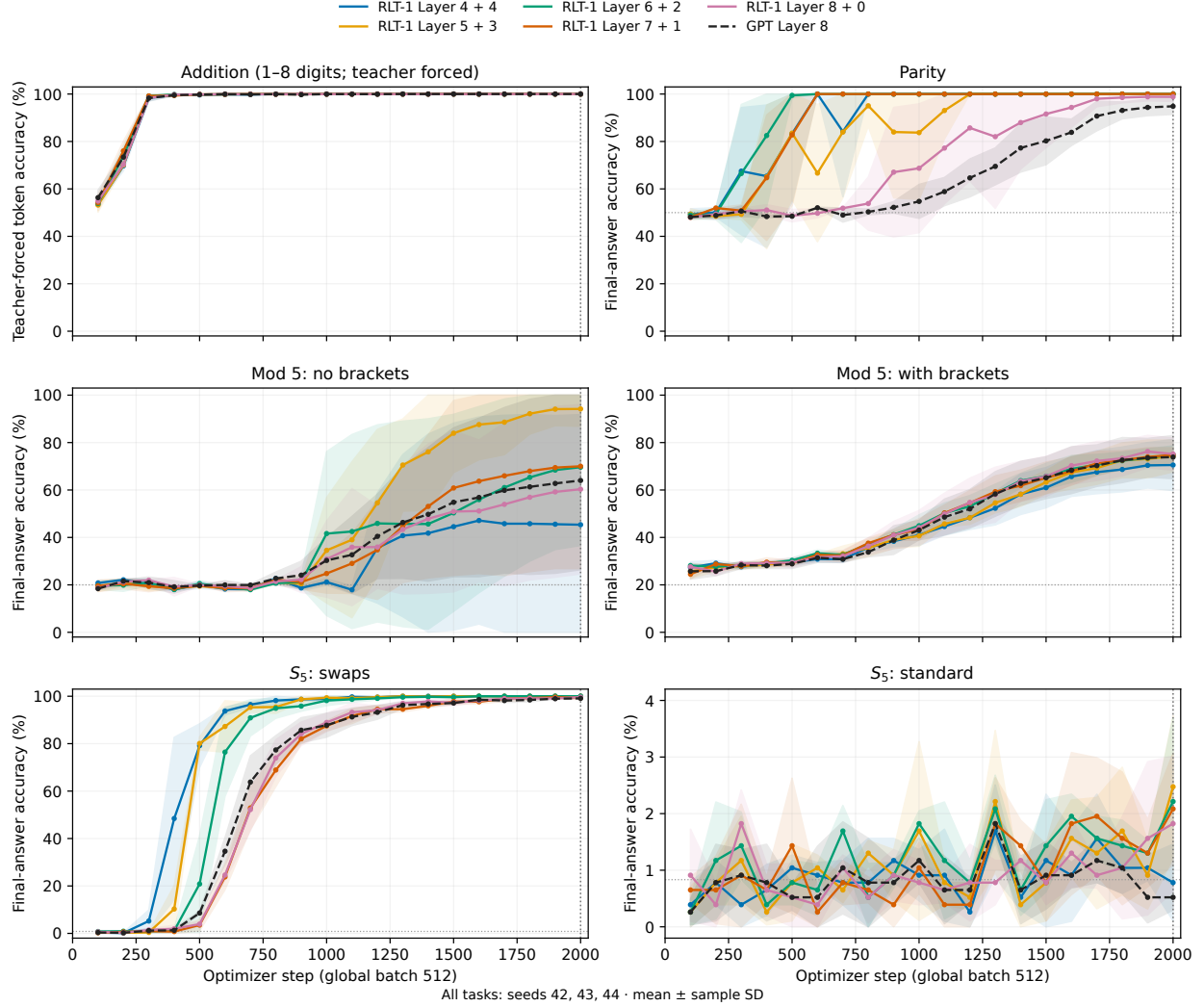


Figure 11 Validation accuracy during training, three seeds for every task. Addition measures teacher-forced answer-token accuracy; the other tasks measure final-label or final-state accuracy. Lines and bands show mean $\pm$ sample SD across seeds 42, 43, and 44 ($n = 3$ at every step); bands are clipped to the accuracy range. Curves are unsmoothed and extend through 2,000 steps. Horizontal dotted lines show uniform-prediction accuracy. Standard S_5 uses a narrower vertical scale.

Bracketed expressions produce closer model means: RLT-1 splits range from 70.53% to 75.17%, compared with $73.87 \pm 9.07\%$ for the Transformer. The two generators differ in operator structure and label distribution. Parentheses also consume positions, so equal token lengths can contain different numbers of arithmetic operations.

J.1.4 Addition and permutation state tracking

All six architectures reach $100 \pm 0\%$ teacher-forced token accuracy on the mixed 1–8-digit addition validation set at step 2,000. On swaps- S_5 , RLT-1 Layer 4 + 4, 5 + 3, and 6 + 2 reach $100 \pm 0\%$ final-state accuracy. The 7 + 1 and 8 + 0 means are $99.35 \pm 0.81\%$ and $99.61 \pm 0.68\%$, compared with $99.09 \pm 0.23\%$ for the Transformer.

On standard S_5 , mean final-state accuracy is 0.52–2.47%, against a uniform 120-class reference of 0.83%. Mean prefix-token accuracy is 5.40–7.99% (Table 5); this metric includes easier early states.

J.1.5 Length generalization

Checkpoint selection follows Section 3. All 18 model–seed combinations for a task receive the same examples at each length: 256 addition pairs or 1,024 formal-task sequences. Figure 12 shows the full grids; Table 3 reports the longest tested lengths. Appendix J.6 gives checkpoint steps and supplementary metrics.

Addition and parity. Teacher-forced addition accuracy falls when operand widths exceed the 1–8-digit training range. At nine digits per operand, RLT-1 Layer 7 + 1 reaches $68.05 \pm 3.64\%$, compared with $58.61 \pm 2.44\%$ for the Transformer. By 32 digits, model means lie between 14.89% and 16.84%.

Parity generalization differs across depth splits even when training-length accuracy is perfect. At 256 bits, 5 + 3 and 7 + 1 retain $100 \pm 0\%$, while 6 + 2 reaches $84.05 \pm 27.63\%$ and 4 + 4 reaches $66.76 \pm 28.78\%$. The 8 + 0 mean is $68.91 \pm 27.39\%$, and the Transformer is near chance at $50.07 \pm 1.63\%$.

Modular arithmetic. On flat expressions of length 63, RLT-1 Layer 5 + 3 reaches $66.96 \pm 40.04\%$, compared with $24.48 \pm 3.89\%$ for the Transformer. At length 127, 5 + 3 retains $39.65 \pm 19.81\%$; the Transformer is at $19.30 \pm 1.04\%$. By length 255, all model means lie between 18.00% and 21.42%, near the 20% uniform reference. The large intermediate-length SDs reflect the different training outcomes seen in the flat mod-5 validation curves. Bracketed mod-5 also loses accuracy as expressions grow longer.

Permutation state tracking. At 256 operations on swaps- S_5 , RLT-1 Layer 4 + 4 reaches $97.30 \pm 2.76\%$ final-state accuracy, compared with $0.85 \pm 0.30\%$ for the Transformer. The 5 + 3 and 6 + 2 means are $92.71 \pm 1.87\%$ and $78.91 \pm 22.26\%$; 7 + 1 and 8 + 0 are near the uniform reference. At 512 operations, 4 + 4 retains $55.70 \pm 25.78\%$ final-state accuracy and $91.16 \pm 6.09\%$ prefix-token accuracy, versus $0.85 \pm 0.30\%$ and $9.33 \pm 0.11\%$ for the Transformer. Standard S_5 remains low at this length, with mean final-state accuracies from 0.72% to 1.43%.

Figure 13 compares three measures of state-tracking accuracy across test lengths. Prefix-token accuracy averages correctness over all intermediate states; final-state accuracy scores only the last state. Whole-sequence accuracy requires every prefix prediction to be correct. A correct final state can follow incorrect intermediate predictions, while prefix-token accuracy can remain high when errors occur late in the sequence.

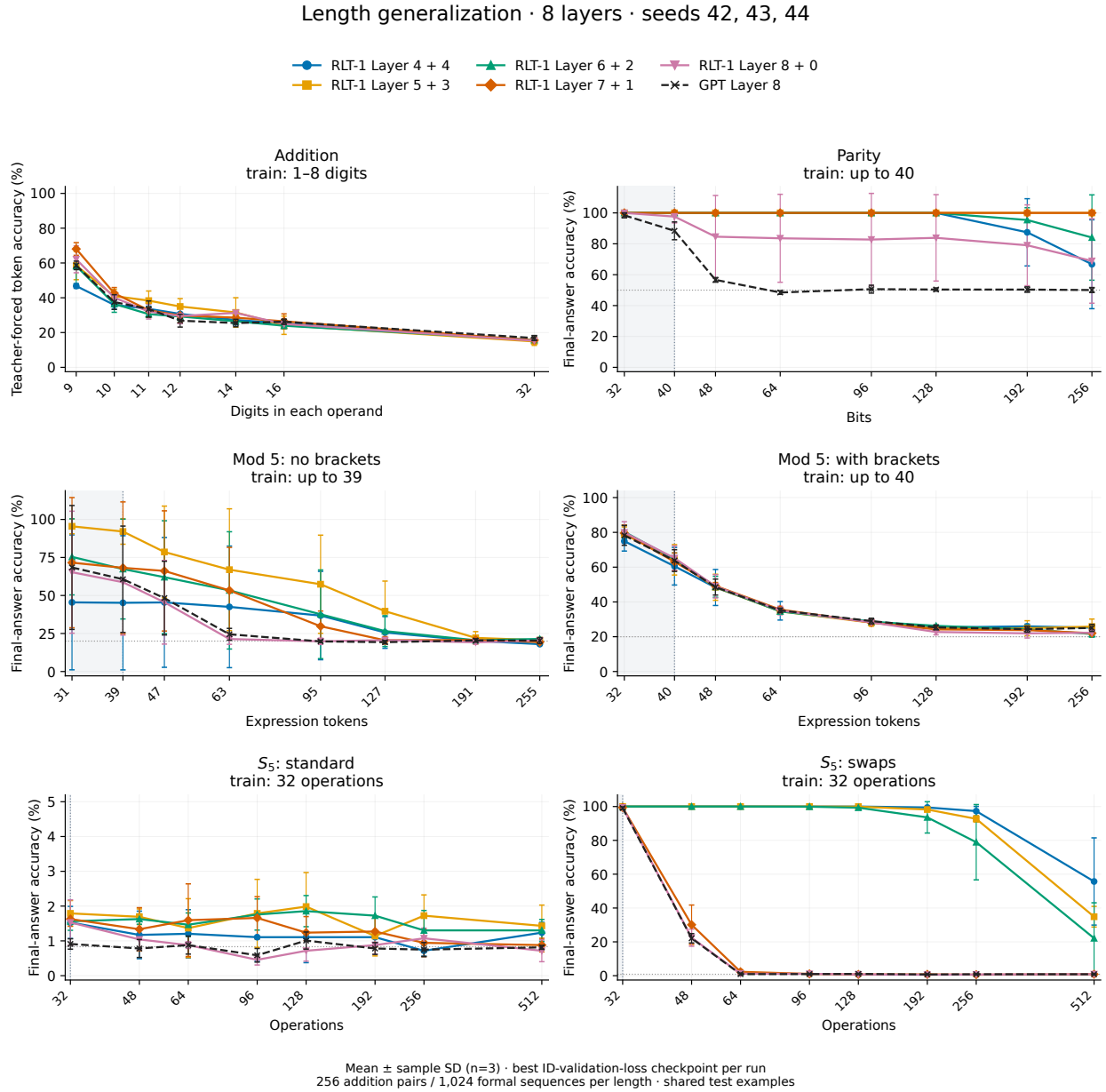


Figure 12 Length generalization of the best in-distribution checkpoints, three initialization seeds. Addition measures teacher-forced answer-token accuracy on 256 pairs per operand width; the other tasks measure final-label or final-state accuracy on 1,024 sequences per length. Points and error bars show mean ± sample SD across seeds 42, 43, and 44, using identical test examples. SD whiskers may extend outside the accuracy range. Gray regions mark trained lengths; horizontal dotted lines show uniform-prediction accuracy. Flat mod-5 uses actual odd expression lengths, excluding BOS and the final equals sign.

S_5 length generalization · three scoring rules

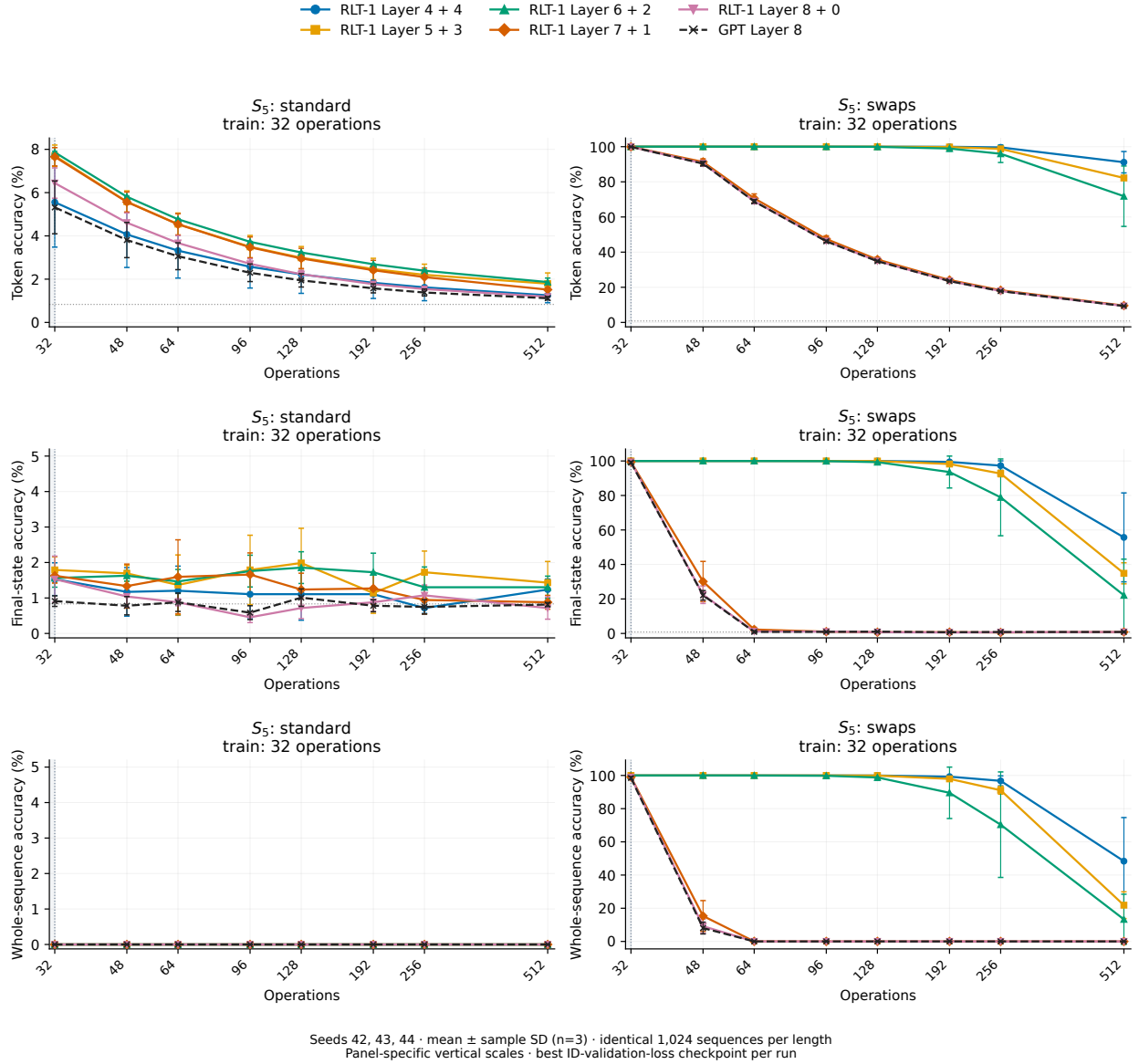


Figure 13 S_5 length generalization under three scoring rules. Rows show prefix-token, final-state, and whole-sequence accuracy; columns show standard and swaps inputs. Every model and seed receives the same 1,024 sequences per length. Points and error bars show mean $\pm$ sample SD across three initialization seeds. Whole-sequence success requires all prefix predictions to be correct. Panel-specific vertical scales expose low accuracies on standard S_5 .

Table 3 Accuracy (%) at the longest tested length. Entries are mean $\pm$ sample SD across three initialization seeds, using the checkpoint and scoring rules of Figure 12. Addition reports teacher-forced token accuracy; formal tasks report final-label or final-state accuracy.

Task (test length)	RLT-1 Layer 4 + 4	RLT-1 Layer 5 + 3	RLT-1 Layer 6 + 2	RLT-1 Layer 7 + 1	RLT-1 Layer 8 + 0	GPT Layer 8
Addition (32)	15.30 $\pm$ 0.44	14.89 $\pm$ 2.27	15.74 $\pm$ 1.58	15.59 $\pm$ 1.57	15.31 $\pm$ 0.93	16.84 $\pm$ 1.45
Parity (256)	66.76 $\pm$ 28.78	100.00 $\pm$ 0.00	84.05 $\pm$ 27.63	100.00 $\pm$ 0.00	68.91 $\pm$ 27.39	50.07 $\pm$ 1.63
Mod 5, no brackets (255)	18.00 $\pm$ 0.39	20.57 $\pm$ 0.62	21.42 $\pm$ 0.49	19.34 $\pm$ 0.54	20.44 $\pm$ 0.91	20.35 $\pm$ 2.01
Mod 5, brackets (256)	25.20 $\pm$ 1.71	25.81 $\pm$ 4.34	21.58 $\pm$ 1.86	22.04 $\pm$ 1.72	22.30 $\pm$ 1.13	25.07 $\pm$ 2.05
S_5 , standard (512)	1.24 $\pm$ 0.30	1.43 $\pm$ 0.60	1.30 $\pm$ 0.31	0.88 $\pm$ 0.20	0.72 $\pm$ 0.31	0.81 $\pm$ 0.06
S_5 , swaps (512)	55.70 $\pm$ 25.78	34.86 $\pm$ 6.10	22.14 $\pm$ 20.95	0.85 $\pm$ 0.06	0.85 $\pm$ 0.20	0.85 $\pm$ 0.30

J.1.6 Training stability and comparison scope

Parity RLT-1 Layer 5 + 3 with seed 42 drops from 100% at step 500 to 48.05% at step 600, then returns to 100% at step 700. Figure 17 shows the unsmoothed losses, and Appendix J gives individual parity curves. This temporary regression and the flat mod-5 seed differences show that final averages can hide unstable learning trajectories.

The comparison with a decoder-only Transformer changes feedback, attention structure, parameter count, and compute together. Appendices J.9 and J.10 isolate the feedback with RLT-0 and RLT-2 at every split.

J.2 Parameter counts

Table 4 lists unique trainable parameters. Each RLT-1 decoder block has both SWA and encoder-memory cross-attention; each Transformer block has one attention sublayer. RLT-1 Layer 8 + 0 applies the learned state projection and gated merge at every token.

Table 4 Unique trainable parameters in the eight-layer models. Each model has the same size across all six tasks.

Model	Parameters (M)	Relative to GPT Layer 8
RLT-1 Layer 4 + 4	28.73	1.135×
RLT-1 Layer 5 + 3	28.20	1.114×
RLT-1 Layer 6 + 2	27.68	1.093×
RLT-1 Layer 7 + 1	27.15	1.073×
RLT-1 Layer 8 + 0	26.10	1.031×
GPT Layer 8	25.31	1.000×

J.3 Implementation of RLT-0

The `rlt_parallel_tiny` control was merged in [nanogptpro-dev PR #164](#). It uses an untied 4 + 4 encoder-decoder split, width 512, four attention heads, FFN width 1,365, vocabulary capacity 277, an SWA window of eight, one encoder-memory group, and width- μ P. It has 27,938,944 unique parameters, and its architecture follows Section 2.7. Training uses full backpropagation, and architecture markers distinguish RLT-0 and RLT-1 checkpoints.

J.4 Seed aggregation and metric definitions

Let $a_{m,s,t}$ be validation accuracy for model m , initialization seed s , and optimizer step t . For every task we report

$$\bar{a}_{m,t} = \frac{1}{3} \sum_{s \in \{42,43,44\}} a_{m,s,t}, \quad \text{SD}_{m,t} = \sqrt{\frac{1}{2} \sum_{s \in \{42,43,44\}} (a_{m,s,t} - \bar{a}_{m,t})^2}. \quad (\text{J.1})$$

We aggregate all three seeds at the same step, without interpolation or imputation. All curves extend through step 2,000, the common comparison step in Table 2.

Parity and mod-5 supervise one final label per example. Their three equally sized validation groups are pooled within a run before averaging across seeds. S_5 supervises each prefix; final-state accuracy counts the last state in each sequence, while prefix-token accuracy pools all supervised positions. Addition uses teacher forcing on the full prompt and correct preceding answer tokens. Its denominator includes all answer tokens—digits, spaces, brackets, and EOS—and excludes prompt and padding positions. Addition loss is teacher-forced cross-entropy normalized per example.

J.5 Reproducing the completed training snapshot

The formal tasks use data seeds 20260914 for training, 20260915 for validation, and 20260916 for test sets. Addition uses data seed 42. All 108 runs completed 2,000 steps using the implementation

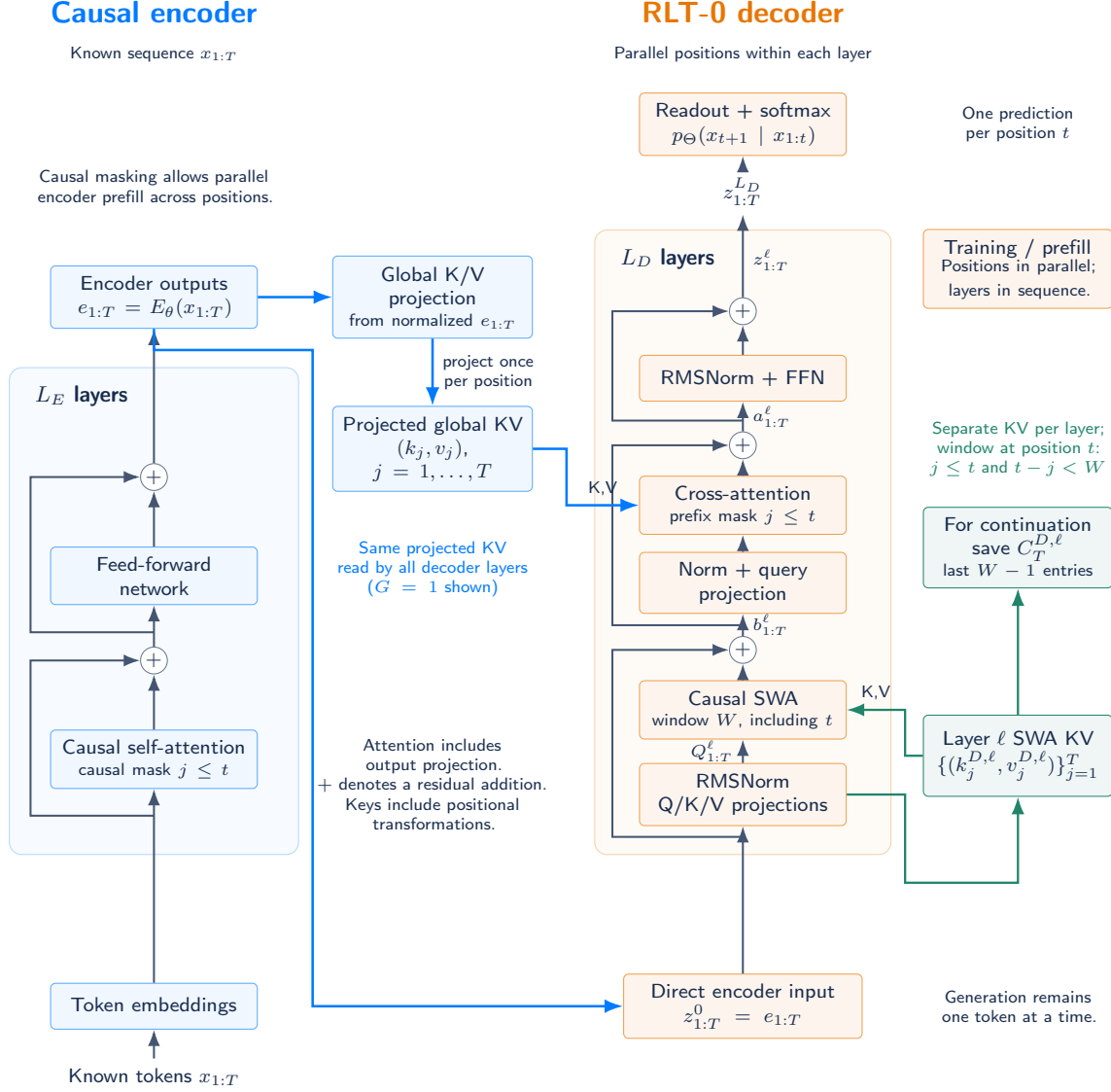


Figure 14 RLT-0 during training or prompt prefill. The decoder receives the encoder outputs directly, $z_{1:T}^0 = e_{1:T}$, with the gated merge and final-hidden-state feedback path removed. Each outlined stack repeats for its indicated depth. For $G = 1$, all decoder layers read the same projected global KV; query position t can read encoder positions $j \leq t$. Each decoder layer constructs its own SWA KV and applies a causal window of size W , including the current position. Known positions run together within each layer, while layers follow depth order. Incremental generation retains the encoder cache, global memory, and each layer’s last $W - 1$ SWA entries and proceeds one token at a time.

Table 5 Final-state and prefix-token accuracy (%) on the fixed S_5 validation sets at step 2,000, reported as mean $\pm$ sample SD over seeds 42, 43, and 44.

Task	Metric	RLT-1 Layer 4 + 4	RLT-1 Layer 5 + 3	RLT-1 Layer 6 + 2	RLT-1 Layer 7 + 1	RLT-1 Layer 8 + 0	GPT Layer 8
S_5 , swaps	Final state	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	99.35 $\pm$ 0.81	99.61 $\pm$ 0.68	99.09 $\pm$ 0.23
S_5 , swaps	Prefix token	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	99.96 $\pm$ 0.05	99.95 $\pm$ 0.06	99.91 $\pm$ 0.06
S_5 , standard	Final state	0.78 $\pm$ 0.68	2.47 $\pm$ 1.26	2.21 $\pm$ 1.48	2.08 $\pm$ 0.98	1.82 $\pm$ 1.19	0.52 $\pm$ 0.23
S_5 , standard	Prefix token	5.57 $\pm$ 2.11	7.82 $\pm$ 0.55	7.99 $\pm$ 0.29	7.67 $\pm$ 0.54	6.47 $\pm$ 0.65	5.40 $\pm$ 1.21

based on upstream commit d1a4516. Native histories were frozen on September 17, 2026; the manifest records exact collection timestamps and source hashes.

manifest.json and configs.json in data/experiments-depth8/ record run identifiers, settings, and source/corpus fingerprints. plot_depth8.py in scripts/ rebuilds training figures and tables from the 2,160 validation records and 216,000 unsmoothed training-step records. The fixed-length token-accuracy export contains 1,800 validation records from 90 completed formal-task runs; plot_token_accuracy.py in the same directory rebuilds its plot.

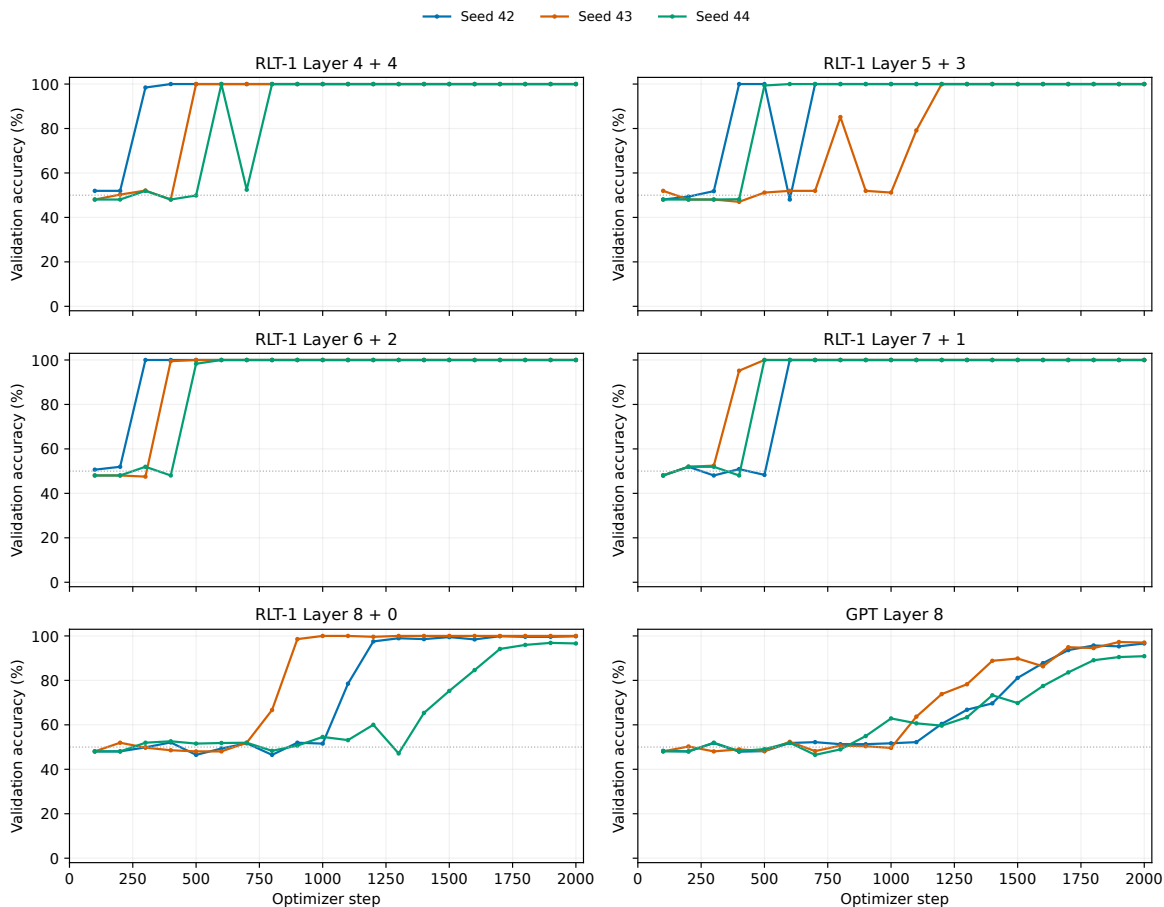


Figure 15 Parity accuracy for each initialization seed through 2,000 steps. All runs share the indexed training stream and validation set; curves show temporary regressions and differences in learning speed.

J.6 Length-generalization protocol and supplementary metrics

The study uses all 108 completed runs, with seeds 42, 43, and 44 for every architecture and task. We select each run’s checkpoint by minimum native ID-validation example loss, pooling the configured formal-task validation lengths and taking the earliest step on ties. Table 6 lists the selected steps; checkpoint and training-result hashes accompany the data. Test results do not participate in selection.

The addition grid is 9, 10, 11, 12, 14, 16, and 32 digits in each operand. Parity and bracketed mod-5 use lengths 32, 40, 48, 64, 96, 128, 192, and 256; both S_5 tasks use 32, 48, 64, 96, 128, 192, 256, and 512. Flat expressions alternate operands and binary operators, so each requested even length L maps to actual expression length $L - 1$: 31, 39, 47, 63, 95, 127, 191, and 255. Formal-task lengths exclude BOS and any final equals sign; flat mod-5 has $L + 1$ input tokens after these markers are added. There are 47 shared test sets and 846 model–seed–length evaluations.

Addition uses 256 unique unordered operand pairs per length, sampled from the native test partition with seed 20260916. Both operands have exactly the indicated width and no leading zero; reversed-digit serialization matches training. The evaluation computes teacher-forced answer-token accuracy without autoregressive generation. Each formal-task set contains 1,024 sequences from the original generator and test seed 20260916. All models and initializations use the same examples at a task and length. Shared holdouts are reused from the previous benchmark; additional formal-task sets exclude duplicates and existing validation/test examples.

After matching checkpoints and holdout fingerprints, we reuse 697 model–seed–length results and evaluate the remaining 149 combinations. All reported metrics are recomputed from saved per-example predictions or teacher-token counts. Evaluation uses CPU FP32, four intra-op threads, and the recorded training runtime; checkpoint weights remain fixed. New evaluations use batch 32, and reused native formal evaluations used batch four. A batch-size check with RLT-1 Layer 4 + 4 matched all 32,768 state predictions across the two batch sizes on 32 length-512 sequences per S_5 task. The teacher-forced addition evaluator also reproduces the saved per-example token counts for RLT-1 Layer 4 + 4 and GPT Layer 8 on 32 shared 32-digit pairs.

At each test length, we first compute a score for each initialization and then apply the mean/SD formula above with length in place of training step. Error bars show sample SD across the three initializations on shared test examples. Figure 16 reports supplementary token accuracy for all six tasks; the three S_5 scoring rules are compared in Appendix J.1.5 (Figure 13).

The directory data/experiments-depth8-generalization/ contains 846 per-seed rows, 282 aggregates, checkpoint/dataset fingerprints, and a verification report. Before regenerating figures and tables, scripts/plot_generalization.py checks seed/length coverage, scores computed from counts, sample SDs, and checkpoint correspondence with the training snapshot.

Length generalization · 8 layers · seeds 42, 43, 44

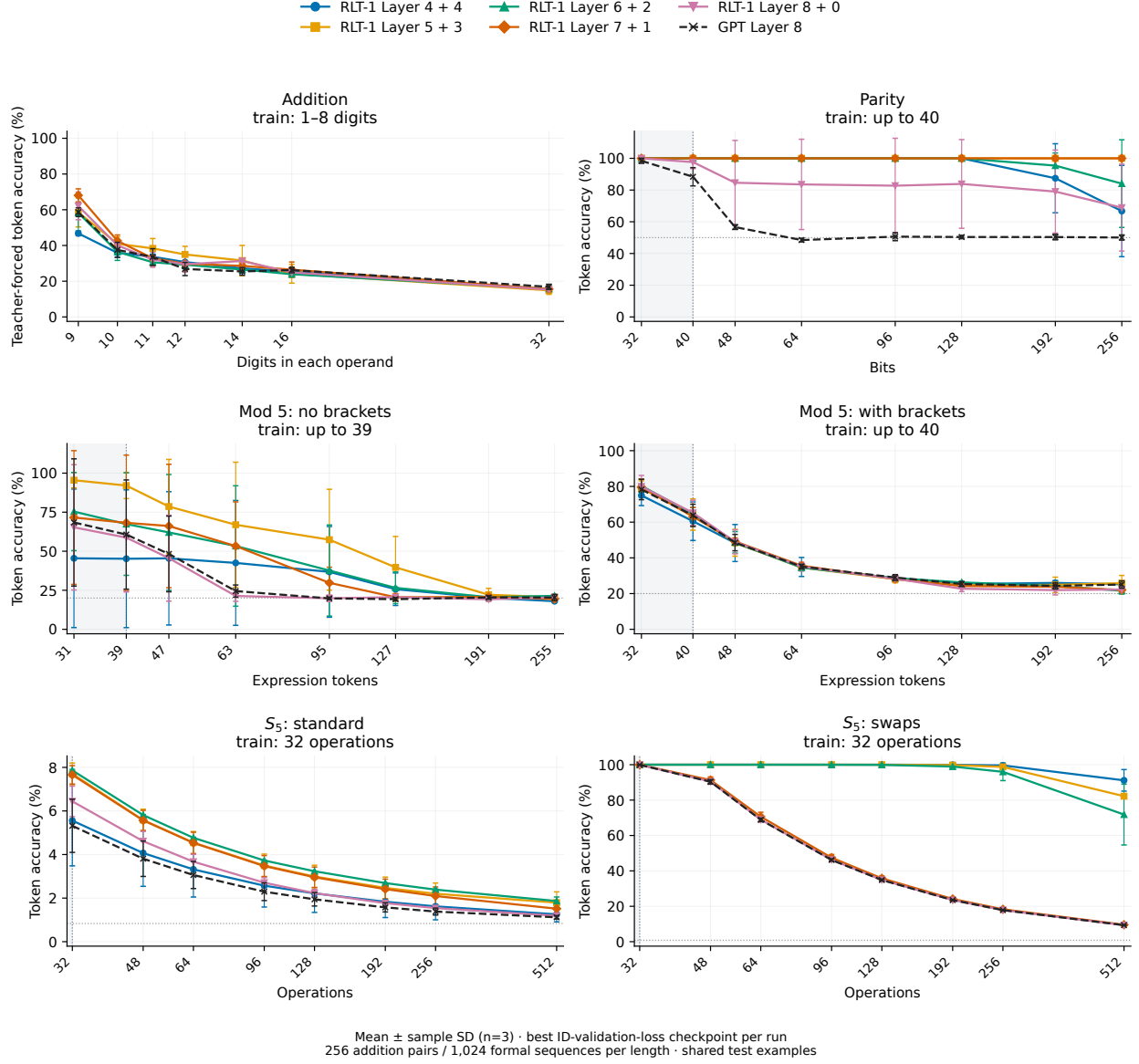


Figure 16 Token accuracy across test lengths, three initialization seeds. The checkpoints and shared examples are those of Figure 12. Addition uses teacher forcing on answer tokens; S_5 scores every prefix state; parity and mod-5 each score one final label. Points and error bars show mean $\pm$ sample SD across seeds 42, 43, and 44. Gray regions mark trained lengths.

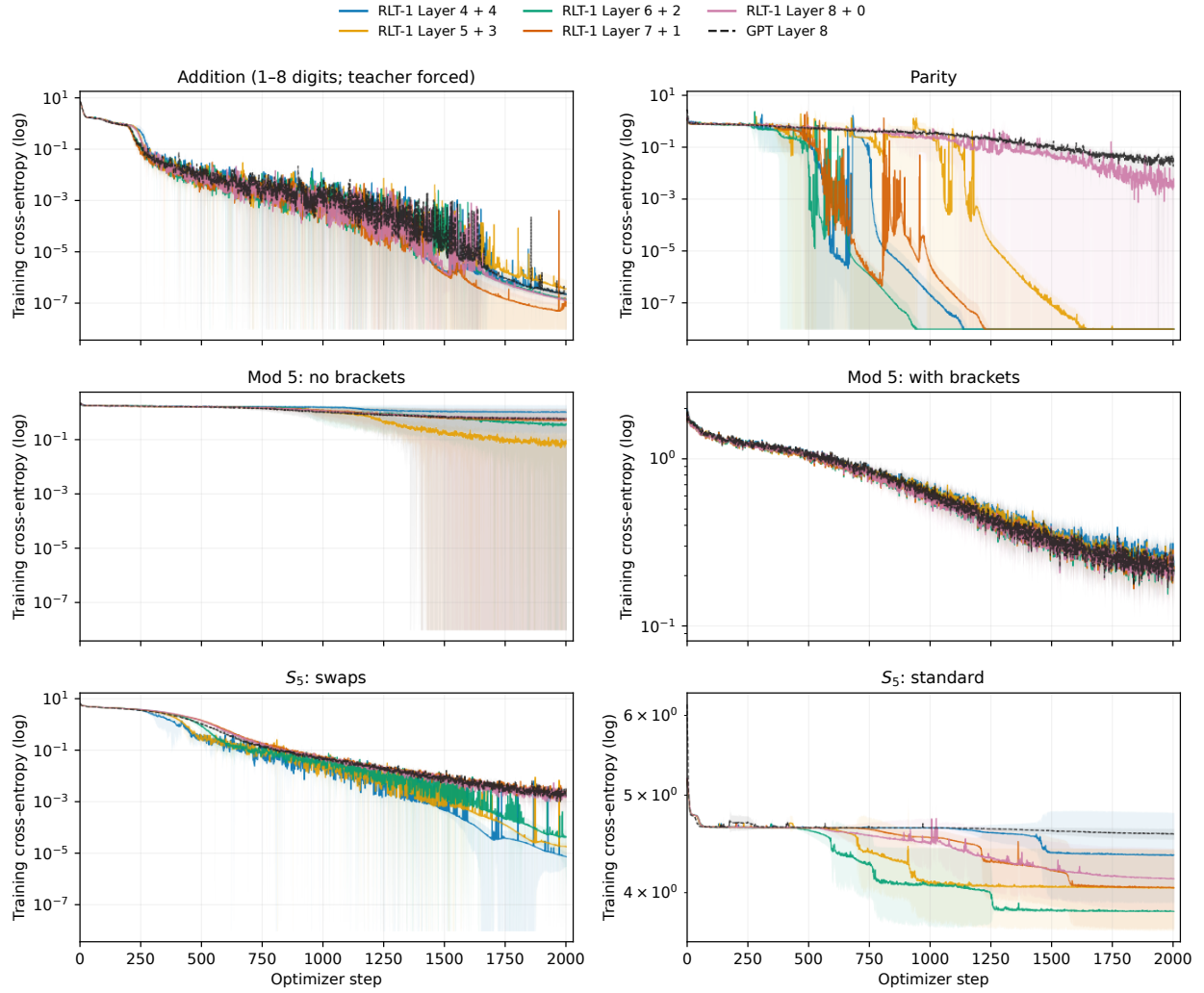


Figure 17 Unsmoothed global-batch training cross-entropy, shown as mean $\pm$ sample SD across three seeds for every task. Means and band limits below 10^{-8} are displayed at 10^{-8} on the logarithmic axis. Loss is measured before the optimizer update; logged gradient norms are measured before clipping. Compare losses within each task, since supervision differs across tasks.

Table 6 Checkpoint steps selected by minimum in-distribution validation loss for length generalization. Every run completed 2,000 steps; seeds 42, 43, and 44 are listed separately.

Task	Seed	RLT-1 Layer 4 + 4	RLT-1 Layer 5 + 3	RLT-1 Layer 6 + 2	RLT-1 Layer 7 + 1	RLT-1 Layer 8 + 0	GPT Layer 8
Addition	42	2000	2000	2000	2000	2000	2000
	43	2000	2000	2000	2000	2000	2000
	44	2000	2000	2000	1900	2000	2000
Parity	42	900	1600	800	1500	2000	2000
	43	1100	2000	900	1500	1900	1900
	44	1400	1000	1200	1100	2000	2000
Mod 5, no brackets	42	800	2000	2000	2000	2000	400
	43	800	2000	2000	800	400	2000
	44	2000	2000	2000	2000	2000	1900
Mod 5, brackets	42	2000	2000	2000	2000	2000	1900
	43	2000	2000	2000	2000	2000	2000
	44	2000	2000	2000	2000	2000	2000
S_5 , standard	42	1800	2000	2000	2000	2000	2000
	43	2000	2000	2000	2000	2000	2000
	44	2000	2000	1800	2000	2000	2000
S_5 , swaps	42	2000	2000	2000	1900	2000	2000
	43	2000	2000	2000	2000	2000	1900
	44	2000	2000	2000	1900	2000	2000

J.7 Token-accuracy trajectories at lengths 32 and 33

Figure 18 compares token accuracy at length 32 for parity, bracketed mod-5, and both S_5 tasks, and at length 33 for flat mod-5. Flat expressions alternate numbers and binary operators, so their payload lengths are odd; 33 is the nearest configured validation length to 32. The completed snapshot contains 1,800 validation checkpoints from 90 runs: six architectures, five formal tasks, and three seeds, each trained for 2,000 steps. At length 32, S_5 token accuracy scores 8,192 prefix states across 256 sequences; parity and mod-5 each score one final label per sequence.

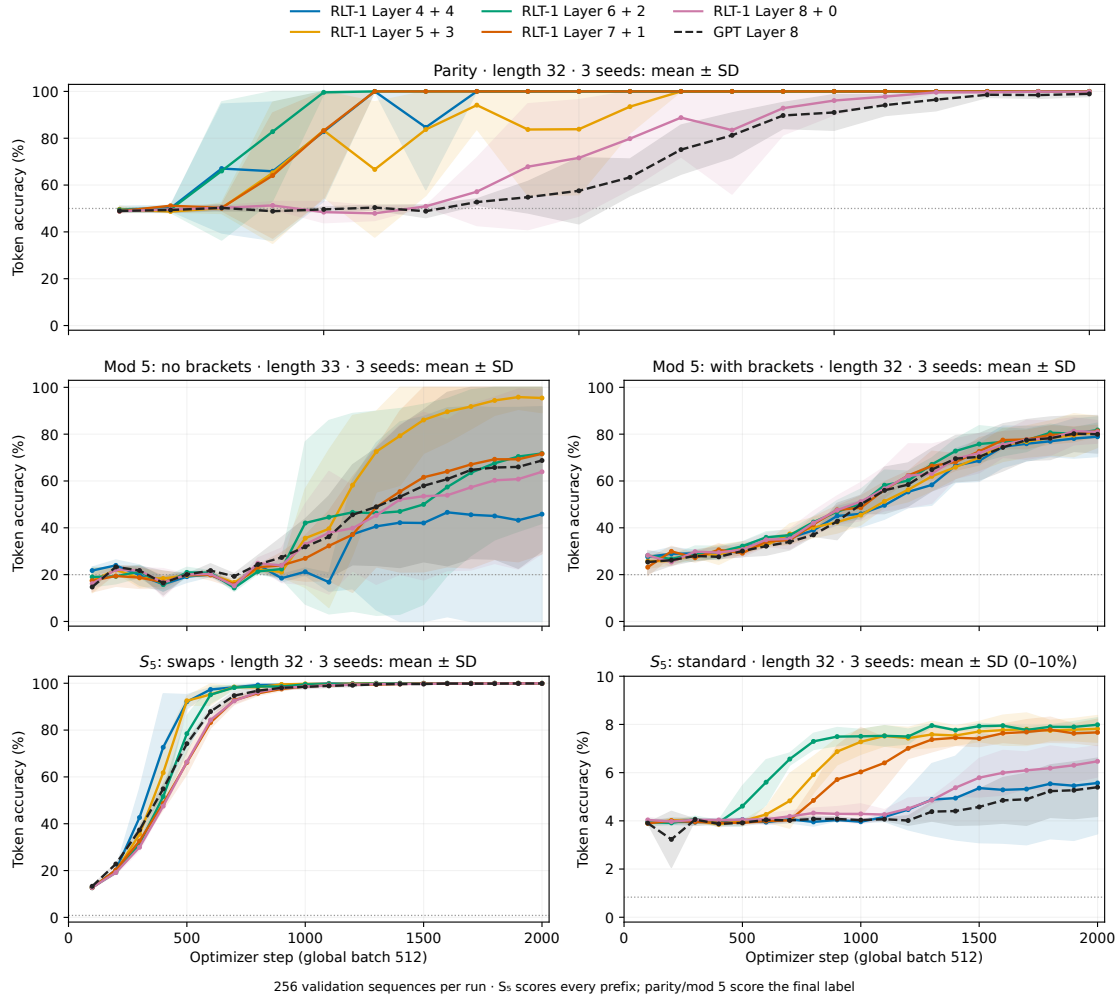


Figure 18 Validation token accuracy on five formal tasks, three seeds per task. Flat mod-5 uses payload length 33; all other panels use length 32. Every point uses all three initialization seeds at the same optimizer step; bands show mean $\pm$ sample SD, clipped to the accuracy range. S_5 scores all prefix states; parity and mod-5 score the final label. All curves extend through 2,000 steps. Standard S_5 uses a narrower vertical scale; horizontal dotted lines show uniform-prediction accuracy.

Table 7 Sixteen-layer parity at 256 bits, seed 42. Final-answer accuracy (%) at the best ID-loss checkpoint and at step 2,000.

Model	Best step	Best checkpoint	Step 2,000
RLT-1 Layer 8 + 8	1400	100.00	100.00
RLT-1 Layer 9 + 7	1300	100.00	100.00
RLT-1 Layer 10 + 6	1800	62.70	66.11
RLT-1 Layer 11 + 5	1300	100.00	100.00
RLT-1 Layer 12 + 4	1000	98.83	98.83
RLT-1 Layer 13 + 3	2000	99.41	99.41
RLT-1 Layer 14 + 2	1500	63.57	62.11
RLT-1 Layer 15 + 1	1100	94.14	94.82
RLT-1 Layer 16 + 0	1900	100.00	100.00
GPT Layer 16	2000	49.41	49.41

J.8 Sixteen-layer parity: best and final checkpoints

All ten seed-42 parity runs completed 2,000 updates with training lengths 3–40, global batch 1,024, and microbatch 32. Figure 19 compares the best ID-loss checkpoint with the fixed step-2,000 checkpoint on the same 1,024 examples per length. Exact ID-loss ties select the earliest checkpoint, giving steps 1,400 and 1,300 for 8 + 8 and 9 + 7. Paired evaluations use matching batch sizes; their best-checkpoint predictions agree with the previously reported predictions.

Accuracy is unchanged in 74 of 80 model–length comparisons. At 256 bits, 10 + 6 changes from 62.70% to 66.11%, 14 + 2 from 63.57% to 62.11%, and 15 + 1 from 94.14% to 94.82%. RLT-1 Layer 8 + 8, 9 + 7, 11 + 5, and 16 + 0 retain 100% at both checkpoints, compared with 49.41% for GPT Layer 16.

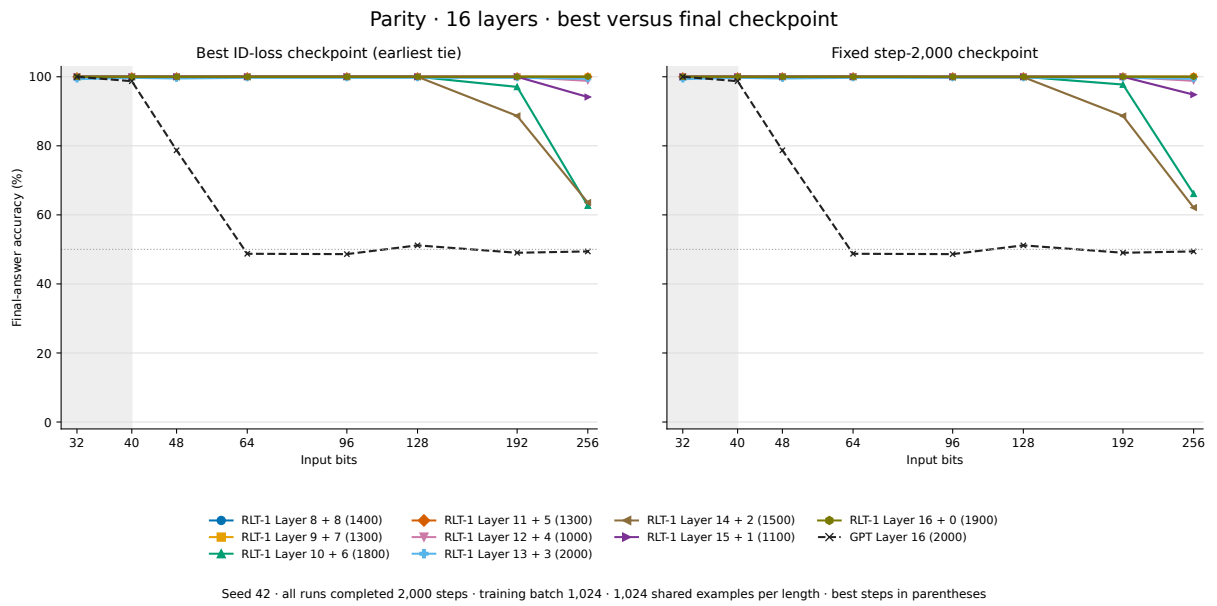


Figure 19 Sixteen-layer parity: best versus final checkpoint. Seed 42, 1,024 shared examples per length; parentheses give best ID-loss checkpoint steps. Gray shading marks the training range; the dotted line marks 50% chance.

Table 8 End-of-training validation for all eight-layer variants. Mean $\pm$ sample SD over seeds 42–44, in percent; addition uses teacher-forced tokens, other tasks use final answers or states. Addition, parity, and S_5 use 2,000 steps; mod-5 uses 5,000.

Family	Layers	Addition	Parity	S_5 swaps	S_5 standard	Mod-5 flat	Mod-5 brackets
RLT-1	4 + 4	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	0.78 $\pm$ 0.68	46.48 $\pm$ 46.35	97.31 $\pm$ 1.14
RLT-1	5 + 3	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	2.47 $\pm$ 1.26	99.13 $\pm$ 1.28	98.26 $\pm$ 0.49
RLT-1	6 + 2	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	2.21 $\pm$ 1.48	99.65 $\pm$ 0.33	98.00 $\pm$ 1.48
RLT-1	7 + 1	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	99.35 $\pm$ 0.81	2.08 $\pm$ 0.98	98.39 $\pm$ 2.78	97.14 $\pm$ 1.16
RLT-1	8 + 0	100.00 $\pm$ 0.00	98.83 $\pm$ 1.92	99.61 $\pm$ 0.68	1.82 $\pm$ 1.19	81.12 $\pm$ 30.90	96.57 $\pm$ 0.85
RLT-0	4 + 4	100.00 $\pm$ 0.00	66.45 $\pm$ 18.12	98.96 $\pm$ 0.23	0.65 $\pm$ 0.81	93.49 $\pm$ 1.92	94.27 $\pm$ 2.00
RLT-0	5 + 3	100.00 $\pm$ 0.00	69.66 $\pm$ 10.15	98.05 $\pm$ 0.68	0.39 $\pm$ 0.39	73.52 $\pm$ 24.88	92.53 $\pm$ 2.76
RLT-0	6 + 2	100.00 $\pm$ 0.00	58.51 $\pm$ 1.50	98.57 $\pm$ 0.60	1.56 $\pm$ 1.17	63.72 $\pm$ 38.67	95.88 $\pm$ 1.64
RLT-0	7 + 1	100.00 $\pm$ 0.00	83.12 $\pm$ 25.82	99.09 $\pm$ 0.81	1.43 $\pm$ 0.60	97.48 $\pm$ 1.01	97.74 $\pm$ 0.53
RLT-0	8 + 0	100.00 $\pm$ 0.00	85.68 $\pm$ 18.25	98.96 $\pm$ 0.81	0.91 $\pm$ 0.60	97.44 $\pm$ 1.73	97.35 $\pm$ 0.67
RLT-2 chunk4	4 + 4	100.00 $\pm$ 0.00	99.87 $\pm$ 0.23	100.00 $\pm$ 0.00	2.47 $\pm$ 1.19	73.87 $\pm$ 44.47	96.40 $\pm$ 1.09
RLT-2 chunk4	5 + 3	100.00 $\pm$ 0.00	83.12 $\pm$ 29.13	99.48 $\pm$ 0.23	1.04 $\pm$ 0.60	71.74 $\pm$ 45.58	98.00 $\pm$ 0.60
RLT-2 chunk4	6 + 2	100.00 $\pm$ 0.00	95.49 $\pm$ 7.82	99.48 $\pm$ 0.23	1.95 $\pm$ 1.70	98.91 $\pm$ 1.33	97.09 $\pm$ 1.66
RLT-2 chunk4	7 + 1	100.00 $\pm$ 0.00	78.43 $\pm$ 16.65	98.83 $\pm$ 1.70	0.65 $\pm$ 0.23	97.70 $\pm$ 2.56	97.01 $\pm$ 1.38
RLT-2 chunk4	8 + 0	100.00 $\pm$ 0.00	76.52 $\pm$ 23.05	99.48 $\pm$ 0.90	0.78 $\pm$ 0.78	98.78 $\pm$ 0.08	96.57 $\pm$ 0.42
GPT	8	100.00 $\pm$ 0.00	94.84 $\pm$ 3.43	99.09 $\pm$ 0.23	0.52 $\pm$ 0.23	99.13 $\pm$ 0.78	97.01 $\pm$ 1.04

J.9 Three-seed comparison of feedback variants

We extend the eight-layer study to RLT-0 and RLT-2 chunk4, each at splits 4 + 4 through 8 + 0 and seeds 42, 43, and 44. These 180 runs train for 5,000 updates on mod-5 and 2,000 on the other four tasks. Width, data generation, optimizer, batch sizes, and CPU execution match Section 3; the cosine schedule ends at the corresponding training budget. RLT-1 and RLT-2 use feedback scale 0.1 and TBPTT128; RLT-0 removes the feedback merge, has 787,968 fewer parameters per split, and uses full BPTT. The three-seed RLT-1 and Transformer controls for addition, parity, and S_5 are reused from the original study.

Table 8 reports validation at the end of training for all 288 runs. Figure 20 shows 4 + 4 training curves; the data repository provides curves for every split. Every validation mean uses seeds 42, 43, and 44 at the same step, including the Transformer mod-5 controls.

Formal-task tests select the minimum ID validation example loss, with earliest-step tie breaking. We recount final-state, prefix-token, and whole-sequence scores from saved predictions and verify identical holdouts across variants and seeds. Each length has 1,024 examples; this comparison uses the native test grid through length 256 (255 for flat mod-5). The separate 512-operation S_5 results in Section 3.2 remain specific to RLT-1 and Transformer.

Section 3 discusses the parity and swaps- S_5 results; Table 9 gives the 4 + 4 values at selected lengths, including both mod-5 tasks. All 30 RLT-0 and RLT-2 addition runs reach 100% teacher-forced token accuracy on mixed 1–8-digit validation at step 2,000. Their length generalization beyond eight digits was not evaluated; the 9–32-digit figures compare RLT-1 with the Transformer.

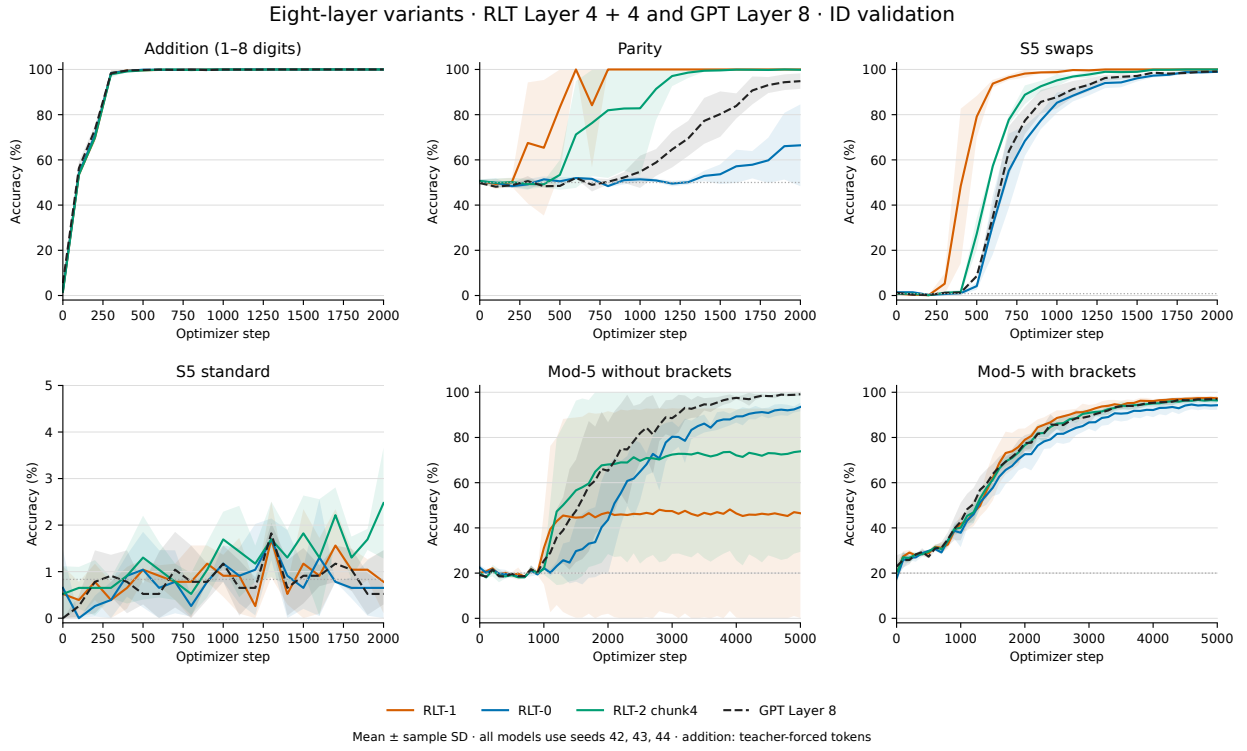


Figure 20 Feedback variants at 4 + 4: validation during training. Lines and bands show mean $\pm$ sample SD over seeds 42–44 for every model. Addition measures teacher-forced answer tokens; other tasks measure final answers or states. Bands are clipped to 0–100%; standard S_5 uses a narrower vertical scale.

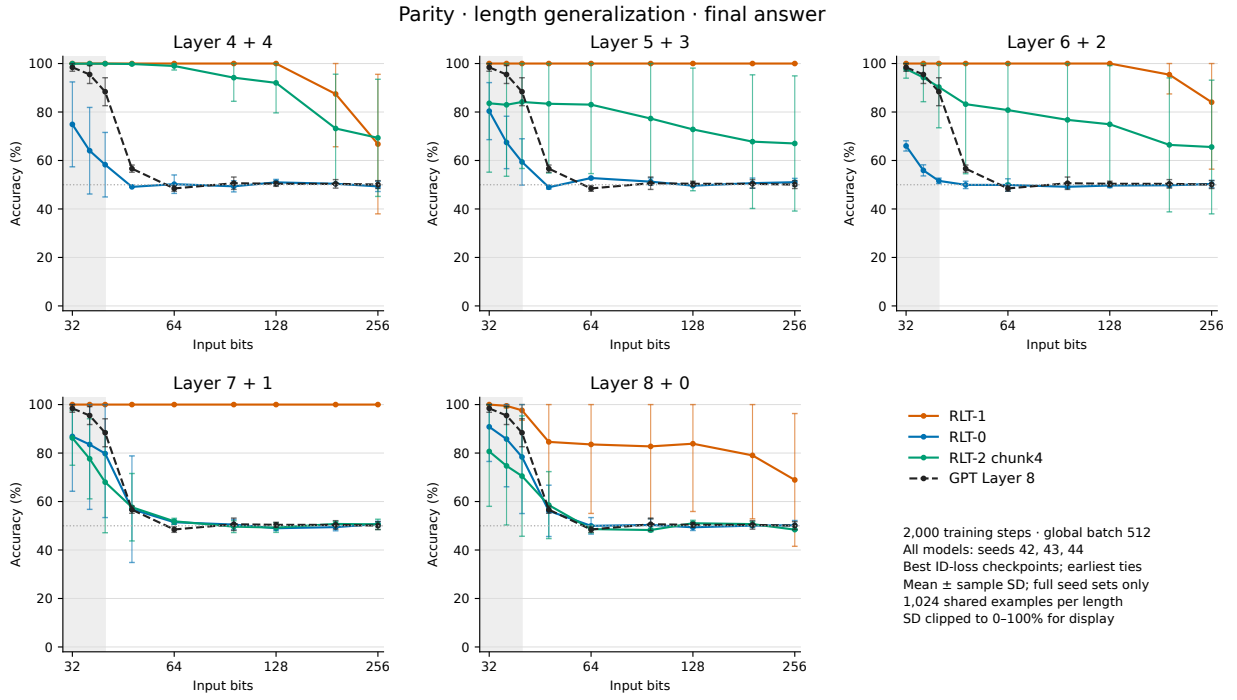


Figure 21 Parity length generalization across feedback variants and depth splits. All models use three seeds, 2,000 training steps, and their best ID-loss checkpoint. Error bars show sample SD, clipped to 0–100% for display; gray shading marks training lengths and the dotted line marks chance.

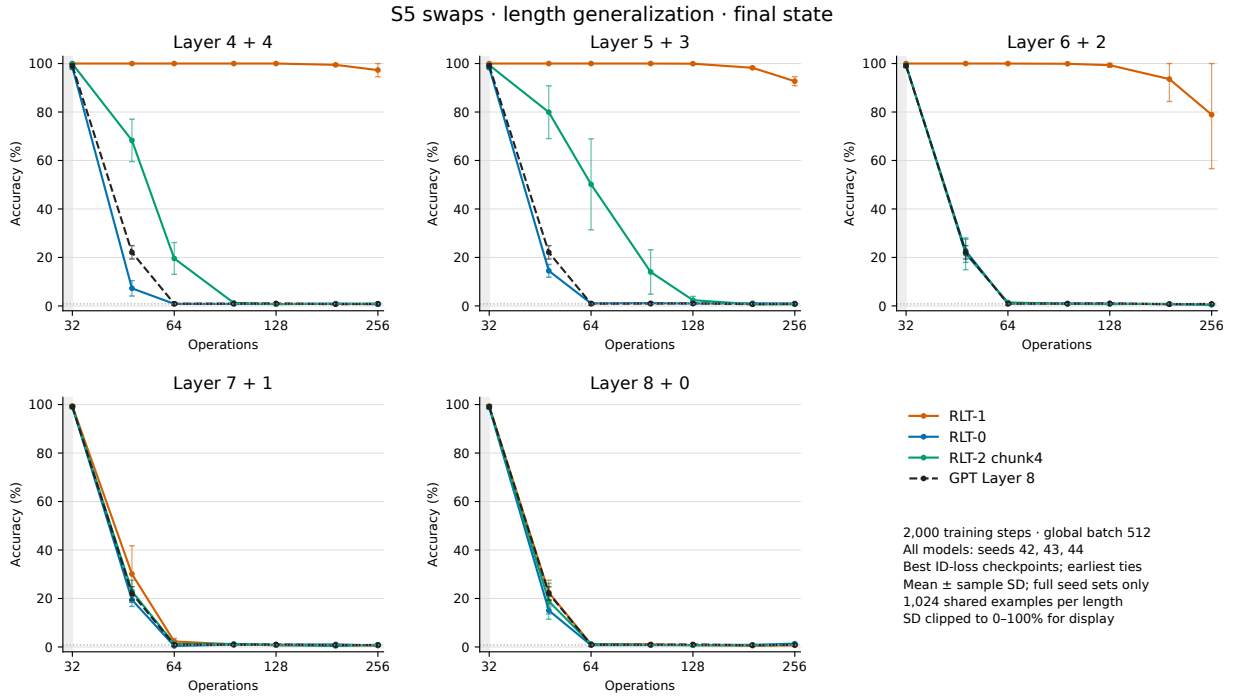


Figure 22 Swaps- S_5 length generalization across feedback variants. Final-state accuracy, mean $\pm$ sample SD over three seeds, using each run’s best ID-loss checkpoint from 2,000 training steps. Error bars are clipped to 0–100% for display.

Table 9 Best-checkpoint test accuracy at 4 + 4. Final-answer/state accuracy (%), mean $\pm$ sample SD over three seeds, on the same 1,024 examples per length. Standard S_5 at length 32 is in distribution; the other rows are out of distribution.

Task / length	RLT-1	RLT-0	RLT-2 chunk4	GPT Layer 8
Parity / 64	100.00 $\pm$ 0.00	50.23 $\pm$ 3.80	98.99 $\pm$ 1.66	48.47 $\pm$ 1.18
S_5 swaps / 48	100.00 $\pm$ 0.00	7.26 $\pm$ 3.17	68.29 $\pm$ 8.75	22.10 $\pm$ 2.76
S_5 standard / 32	1.53 $\pm$ 0.46	1.20 $\pm$ 0.72	1.37 $\pm$ 0.87	0.91 $\pm$ 0.15
Mod-5 flat / 63	44.43 $\pm$ 43.81	21.84 $\pm$ 2.17	50.78 $\pm$ 38.02	33.20 $\pm$ 2.33
Mod-5 brackets / 64	64.10 $\pm$ 3.49	39.45 $\pm$ 0.61	62.76 $\pm$ 4.42	46.71 $\pm$ 1.21

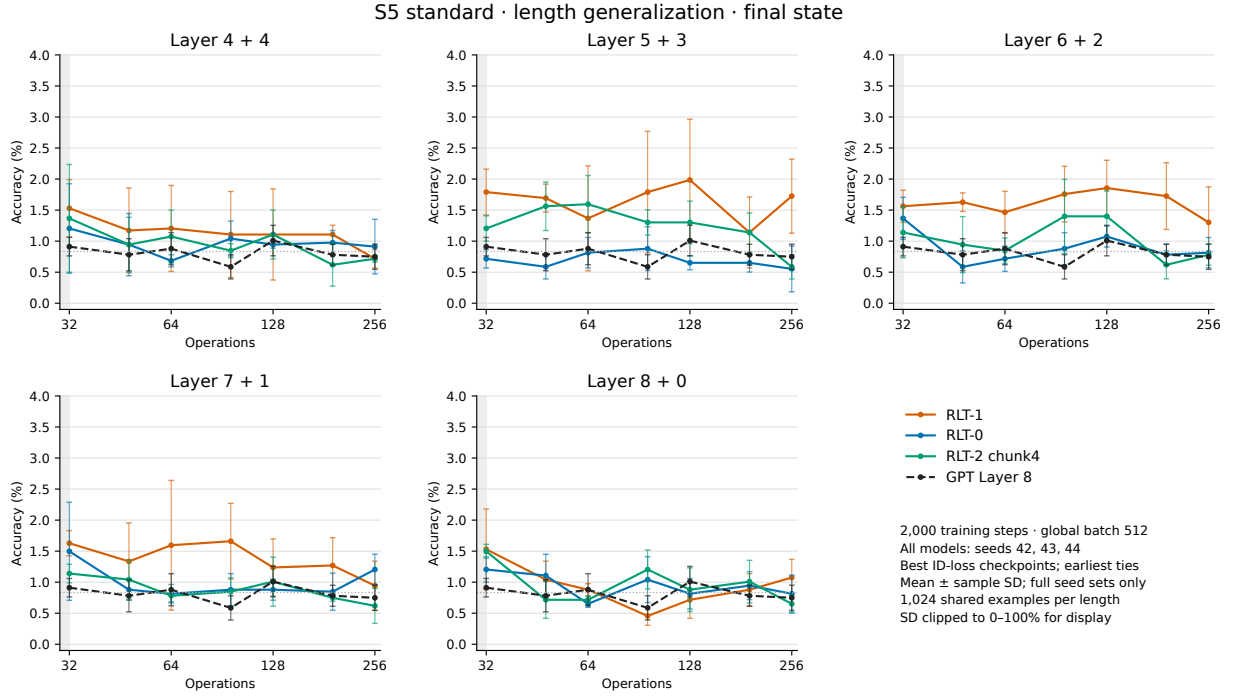


Figure 23 Standard S_5 length generalization across feedback variants. Final-state accuracy, mean $\pm$ sample SD over three seeds; all models train for 2,000 steps. The vertical axis is expanded near chance (1/120).

J.10 Mod-5 feedback variants at 5,000 steps

All 96 mod-5 runs train for 5,000 updates. RLT-0, RLT-1, and RLT-2 chunk4 each contribute 30 runs across two tasks, five splits, and seeds 42–44. The Transformer contributes six runs, with the same three seeds on both tasks.

Flat mod-5 remains sensitive to initialization after 5,000 steps. For chunk4 4 + 4, best-checkpoint ID accuracy at length 33 is 100.00%, 20.21%, and 98.73% for seeds 42, 43, and 44. At length 63, its three-seed mean is $50.78 \pm 38.02\%$, compared with $44.43 \pm 43.81\%$ for RLT-1, $21.84 \pm 2.17\%$ for RLT-0, and $33.20 \pm 2.33\%$ for GPT Layer 8.

On bracketed mod-5 at length 64, RLT-1 Layer 4 + 4 reaches $64.10 \pm 3.49\%$, chunk4 $62.76 \pm 4.42\%$, RLT-0 $39.45 \pm 0.61\%$, and GPT Layer 8 $46.71 \pm 1.21\%$.

In the seed-42 CPU timing comparison at 4 + 4, chunk4 training steps are 2.27 times as fast as RLT-1, and RLT-0 steps 4.17–4.30 times as fast, across the two mod-5 tasks. These single-seed means cover steps 1,001–2,000 on the shared cluster and exclude validation, checkpoint writes, and logging. Frozen scores, per-seed records, provenance, and figure regeneration instructions are in `data/variants-20260925/`.

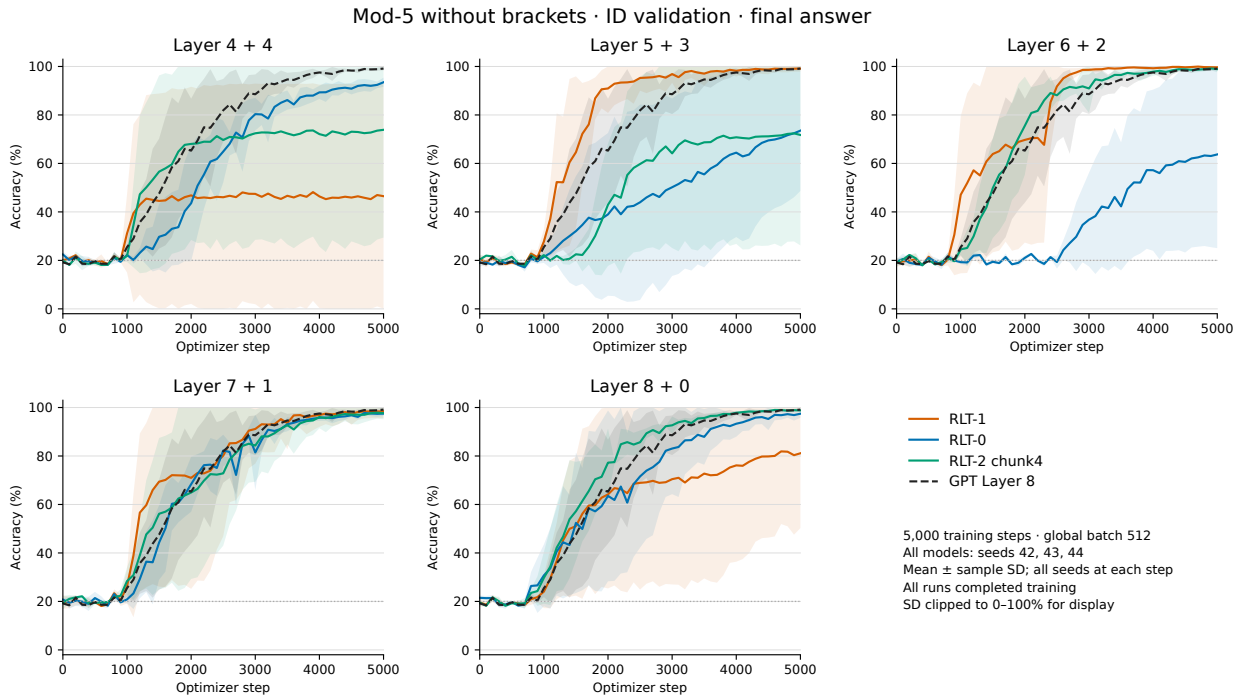


Figure 24 Mod-5 without brackets: ID validation. Mean $\pm$ sample SD over seeds 42–44 for every model; all runs train for 5,000 steps. Bands are clipped to 0–100%.

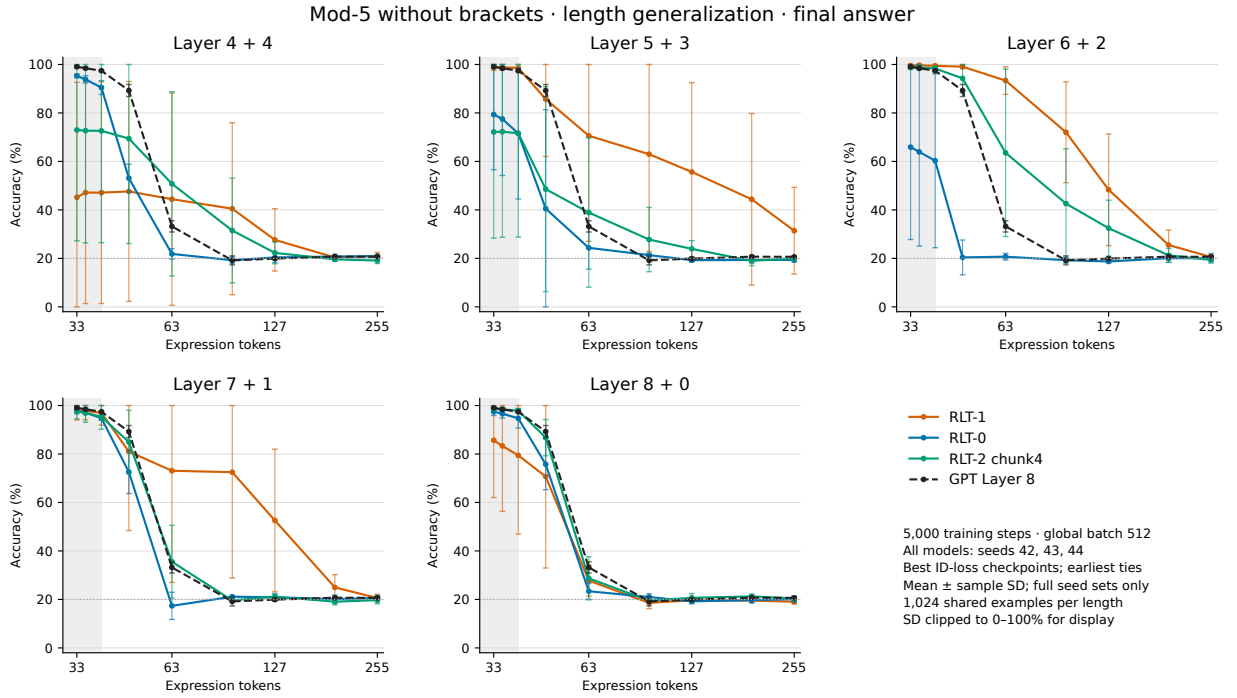


Figure 25 Mod-5 without brackets: length generalization. 5,000-step runs at best ID-loss checkpoints; every curve shows mean $\pm$ sample SD over seeds 42–44. Error bars are clipped to 0–100% for display.

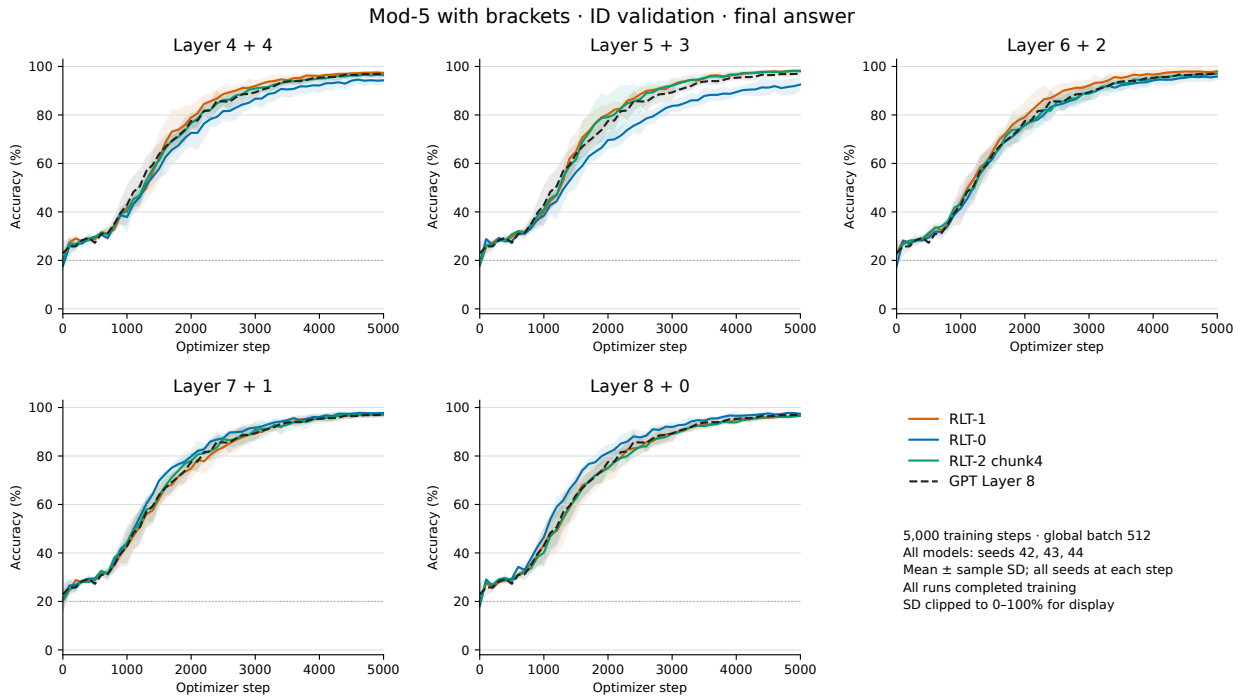


Figure 26 Mod-5 with brackets: ID validation. Mean $\pm$ sample SD over seeds 42–44 for every model; all runs train for 5,000 steps. Bands are clipped to 0–100%.

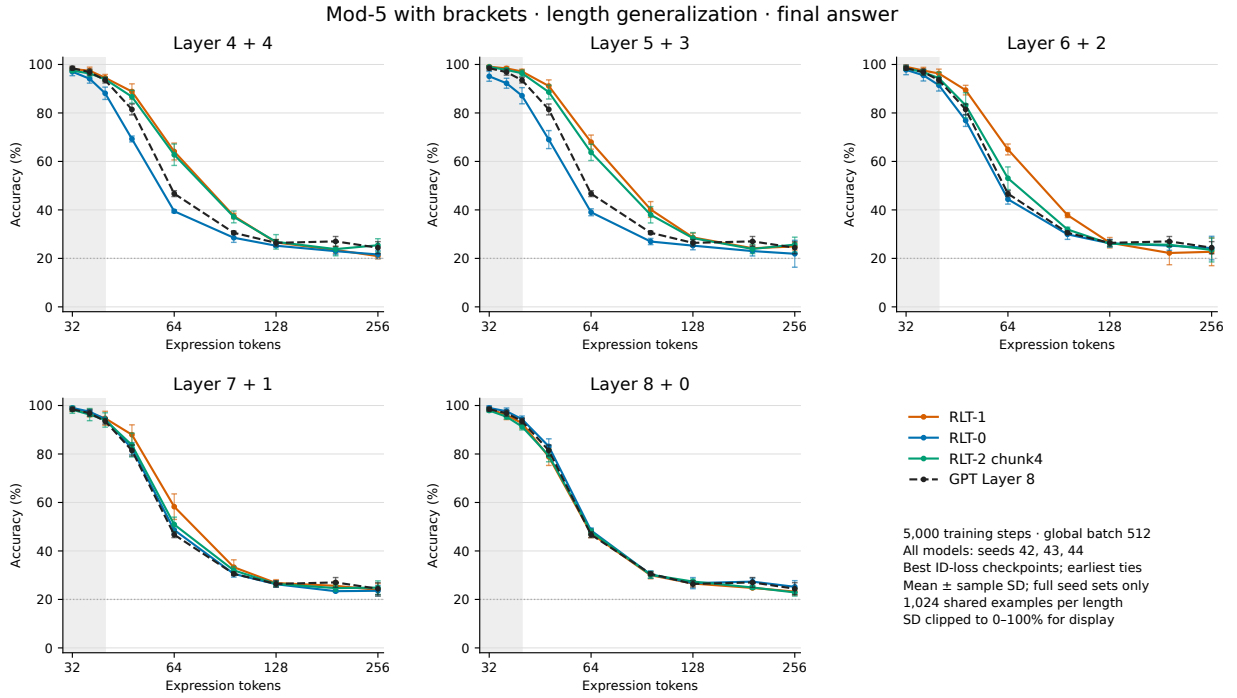


Figure 27 Mod-5 with brackets: length generalization. 5,000-step runs at best ID-loss checkpoints; every curve shows mean $\pm$ sample SD over seeds 42–44. Error bars are clipped to 0–100% for display.

J.11 Addition feedback scale at a fixed learning-rate schedule

We vary only the RLT-1 feedback scale α , keeping the initialization, architecture, training data, and optimizer schedule fixed within each split and seed. All runs train on mixed 1–8-digit addition for 2,000 updates, with global batch 512, microbatch 32, and TBPTT128. The learning rate warms up for 200 steps to 10^{-4} and decays to 5×10^{-6} . We test $\alpha = 0.03$ at every split and $\alpha = 0.01$ at every split except $7 + 1$, with seeds 42, 43, and 44. The comparison includes the $\alpha = 0.1$ RLT-1 runs and GPT Layer 8 at the same seeds, for 45 models in total.

We select each model’s checkpoint by minimum ID validation loss, taking the earliest step on exact ties; the $7 + 1$ run with $\alpha = 0.1$ and seed 44 selects step 1,900, and the other 44 models select step 2,000. Figure 28 and Table 10 report teacher-forced answer-token accuracy on 256 shared examples at each of seven widths from 9 to 32 digits, as mean $\pm$ sample SD over the three seeds. Counts include answer formatting and EOS, and exclude prompts and padding.

No tested feedback reduction improves accuracy at every width for any split. Across the 63 comparisons with $\alpha = 0.1$ at the same split and width, 37 means rise and 26 fall, by -3.86 to $+5.60$ percentage points; only three differences exceed both sample SDs. For $8 + 0$, $\alpha = 0.01$ raises the 12-digit mean from $29.49 \pm 4.38\%$ to $32.21 \pm 5.13\%$, while the 16-digit mean moves from $24.92 \pm 0.39\%$ to $24.66 \pm 4.88\%$. At 32 digits, RLT-1 means range from 14.44% to 15.91%, compared with $16.84 \pm 1.45\%$ for GPT Layer 8.

Table 10 Addition feedback-scale ablation, seeds 42–44. Teacher-forced answer-token accuracy (%) at selected widths, mean $\pm$ sample SD over three seeds, after 2,000 training steps with the same learning-rate schedule. Each width uses 256 shared examples; all seven widths are available in the data repository.

Model	α	9 digits	12 digits	16 digits	32 digits
RLT-1 Layer 4 + 4	0.01	47.25 ± 4.04	33.11 ± 4.56	25.93 ± 1.88	14.44 ± 0.17
RLT-1 Layer 4 + 4	0.03	48.75 ± 5.34	32.00 ± 2.07	26.22 ± 0.74	14.74 ± 1.25
RLT-1 Layer 4 + 4	0.1	46.77 ± 1.34	30.72 ± 0.58	25.28 ± 1.48	15.30 ± 0.44
RLT-1 Layer 5 + 3	0.01	57.96 ± 0.67	32.36 ± 2.93	23.01 ± 5.22	15.29 ± 2.19
RLT-1 Layer 5 + 3	0.03	56.30 ± 2.41	32.60 ± 5.61	23.54 ± 3.75	14.66 ± 1.49
RLT-1 Layer 5 + 3	0.1	58.51 ± 8.12	34.97 ± 4.54	24.20 ± 5.26	14.89 ± 2.27
RLT-1 Layer 6 + 2	0.01	56.52 ± 9.87	30.97 ± 4.64	26.35 ± 2.98	15.21 ± 1.46
RLT-1 Layer 6 + 2	0.03	63.85 ± 10.15	30.39 ± 2.85	27.18 ± 4.26	14.86 ± 0.65
RLT-1 Layer 6 + 2	0.1	58.25 ± 10.29	29.22 ± 2.28	23.92 ± 1.58	15.74 ± 1.58
RLT-1 Layer 7 + 1	0.03	66.43 ± 2.53	30.10 ± 2.29	26.27 ± 1.42	15.91 ± 0.63
RLT-1 Layer 7 + 1	0.1	68.05 ± 3.64	29.80 ± 4.03	26.45 ± 4.31	15.59 ± 1.57
RLT-1 Layer 8 + 0	0.01	62.20 ± 9.85	32.21 ± 5.13	24.66 ± 4.88	15.45 ± 0.66
RLT-1 Layer 8 + 0	0.03	59.18 ± 4.47	28.81 ± 3.57	25.92 ± 1.10	14.90 ± 0.54
RLT-1 Layer 8 + 0	0.1	62.13 ± 7.75	29.49 ± 4.38	24.92 ± 0.39	15.31 ± 0.93
GPT Layer 8	–	58.61 ± 2.44	26.84 ± 3.69	26.22 ± 1.54	16.84 ± 1.45

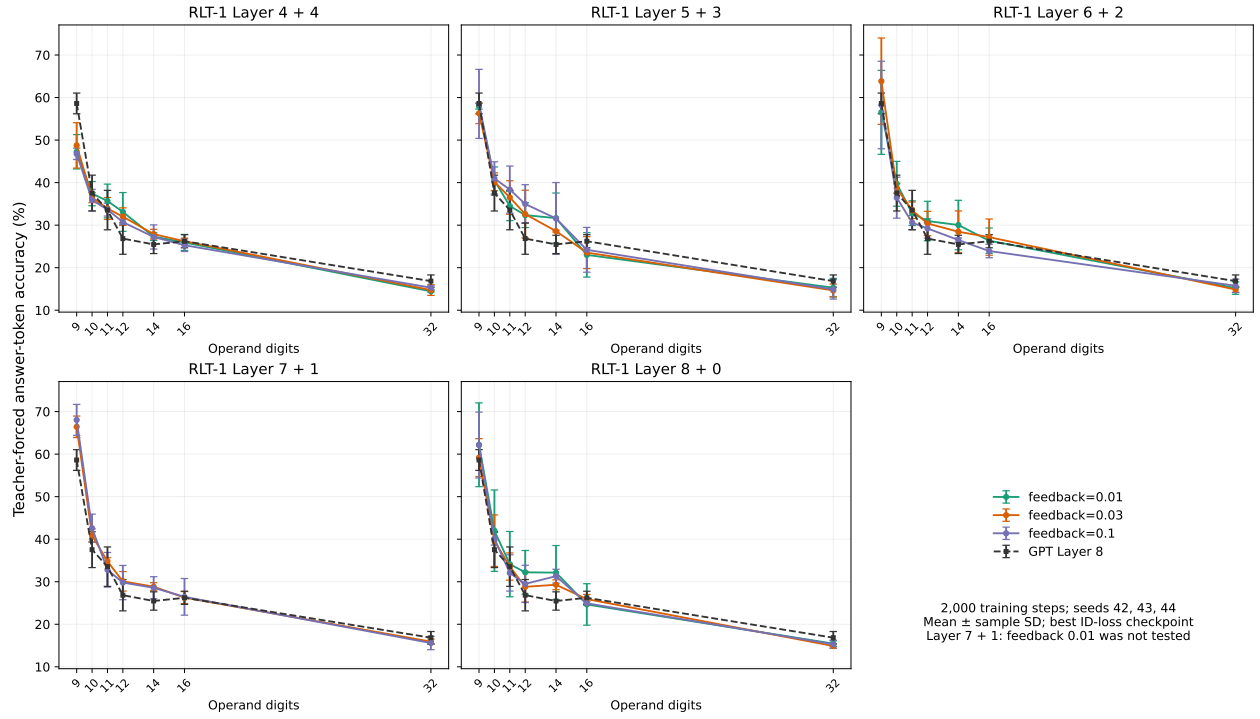


Figure 28 Addition generalization as feedback scale changes. Each panel compares the tested feedback scales with GPT Layer 8 at one RLT-1 split; $\alpha = 0.01$ was not tested at 7 + 1. Points and error bars show mean $\pm$ sample SD over seeds 42, 43, and 44; all models use 2,000 training steps and the same learning-rate schedule. Accuracy measures teacher-forced answer tokens on the same 256 examples at each width.