FULL-BANDWIDTH TRANSFORMER

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoregressive transformers compute along two axes: horizontally across generated tokens, and vertically through model depth. Dense attention gives each token broad horizontal access to the past, but the vertical feedback channel between decoding steps remains narrow: only the sampled token returns to the bottom of the stack, while the top-layer hidden state is discarded. We introduce the *full-bandwidth transformer*, which widens this channel with *latent feedback*: at each decoding step, the previous top-layer hidden state is fused with the sampled token embedding through a gated linear unit and fed back as the next input. Latent feedback lets non-verbalized computation re-enter the stack with a renewed depth budget, while preserving the standard transformer architecture, KV cache, and language-modeling objective. To train full-bandwidth transformers without losing parallel teacher forcing, we use a scheduled multi-pass objective that introduces latent feedback late in pretraining and mixes a small fraction of deeper feedback passes for stability. We train 1B-parameter full-bandwidth transformers up to 400B tokens and find that latent feedback improves validation loss, 5-shot language-model evaluation, math and coding generation, and instruction-tuned performance. With negligible per-token decoding overhead, full-bandwidth transformers match or approach standard transformers trained with roughly $1.5\times$ more tokens, and manage to produce shorter reasoning when no off-policy templates are provided.

1 INTRODUCTION

Autoregressive transformers contain a feedback loop: the token sampled at step $t-1$ becomes the input at step t (Fig. 1, left). This loop is what lets chain-of-thought decoding (Wei et al., 2022) perform computation whose depth grows with the number of generated tokens (Li et al., 2024b). But measured as a communication channel, the loop is narrow: Decoding compresses the model's entire top-layer state, a D-dimensional vector, down to a single symbol carrying at most $\log_2 |V|$ bits. Non-verbalized computation is not erased, intermediate activations persist in the KV cache, but it is *depth-frozen*: a state produced at layer ℓ is readable only by layers above ℓ, so it can never return to the bottom of the stack for further processing, and the deepest, top layer's output is never cached. Verbalization is thus the only channel for information to re-enter the bottom layer and receive fresh computation, at the cost of being squeezed through a single token, forcing the model either to spend tokens narrating its intermediate state or recompute that state from scratch at every position.

In this work, we propose *full-bandwidth transformer* where we widen this channel to its full width. In particular, we introduce latent feedback decoding, which fuses the previous top-layer hidden state with the sampled token's embedding during decoding, and feeds the result back as the next input (Fig. 1 right, Sec. 3.1). We call a transformer capable of decoding this way a full-bandwidth transformer, since its inter-step feedback now carries the entire hidden state rather than a thin token. The sampled token is retained, so the model still produces ordinary text and can be flexibly trained with standard supervised language modeling losses; what changes is that the feedback is no longer limited to the token's identity. By design, this affords two things standard decoding lacks: (i) non-verbalized state can re-enter the bottom of the stack with a renewed depth budget and be processed further across steps, rather than staying frozen in the cache at the level where it was produced; (ii) every layer, including the shallowest, sees the past as processed by the *full* stack, not only by the layers beneath it; Crucially, these come with almost no architecture changes and extra serving cost: the fusion adds two matrix multiplications per generated token, attention and the KV cache are untouched, and prefill is run either once or, optionally, twice for better performance.

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

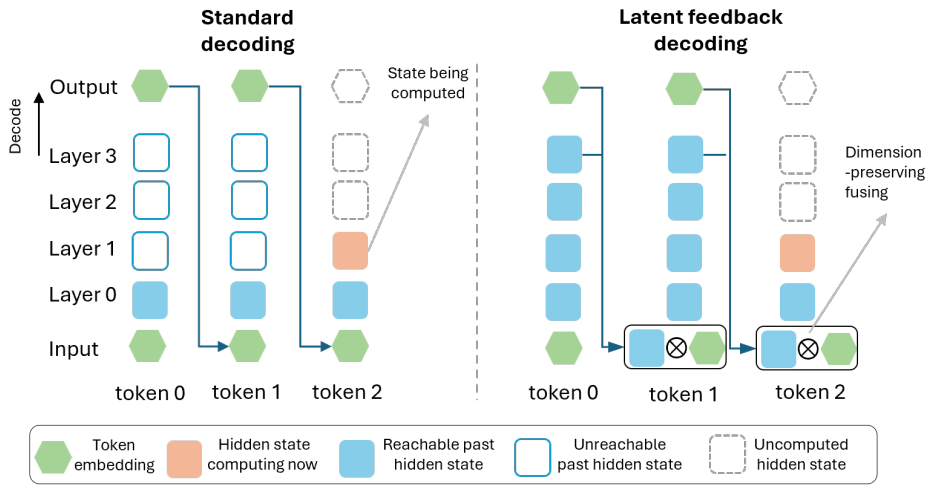


Figure 1: **Standard decoding vs. latent feedback decoding. Left**: In a standard transformer, the current state can access only lower-layer past states (blue); deeper past states (white) are unreachable, and the only inter-step feedback is the sampled token embedding (green). **Right**: A *full-bandwidth transformer* uses *latent feedback*, fusing the previous top-layer hidden state with the sampled token embedding through a dimension-preserving gate ($\otimes$, Eq. (4)) and feeding it back as the next input. This returns full hidden-state information to the bottom of the stack, making the past as processed by all layers accessible to subsequent computation.

The obstacle is training. A pretrained model has never seen hidden states in its input, so latent feedback cannot simply be switched on at inference; and the recurrence it defines is sequential over positions, so training on it directly would forfeit the parallel teacher forcing that makes transformers efficient to train. We resolve this with a *multi-pass* regime (Sec. 3.2): each pass shifts the previous pass's hidden states one position rightward, fuses them with the token embeddings, and re-runs the stack in parallel across all positions, so sequentiality is paid across multiple forward passes rather than across the sequence. To make this practical at large scale, we adopt a *progressive schedule*, spending the bulk of training (e.g. first 75%) on the ordinary single-pass objective such that the run can start from a standard pretraining checkpoint and introduces extra feedback passes only late; Empirically, we find the schedule's composition matters in an unexpected way: training with two feedback passes alone produces a recurrence that *diverges* once rolled past its trained depth, whereas mixing in as little as 3% three-pass batches turns the learned map into a stable attractor, which stays stable beyond the trained depth (Fig. 3).

Empirically, full-bandwidth transformers show performance equivalent to substantially more training data. Utilizing two or three forward passes for prefill, the model matches standard transformer baselines trained on twice the tokens in both validation loss and multiple-choice accuracy (Fig. 4). On free-form generation (Fig. 5)—GSM8K,HumanEval, MBPP—latent feedback improves over standard decoding of under the *same* weights, matches the $2\times$-token baselines, and on some tasks approaches baselines trained with up to $5\times$ the tokens; the gains carry over through long-context extension and instruction tuning (Table 1). Additionally, latent feedback often yields shorter reasoning traces, (Fig. 6 and 20), the behavior the widened channel predicts, with compute state stored in the latents instead of being verbalized by tokens.

2 Background

Given a vocabulary of size $|V|$ and a D-dimensional residual stream, a decoder-only LLM, $f_\theta(\cdot)$, maps an input sequence of T tokens, with embeddings $\{\boldsymbol{e}_t\}_{t=1}^{T} \in \mathbb{R}^{T\times D}$, through L attention–MLP blocks. The final-layer hidden states $\{\boldsymbol{h}_t^L\}_{t=1}^{T}$ are then projected and decoded into the next-token

$$\boldsymbol{h}_t^L = f_\theta(\boldsymbol{e}_t;\, C), \qquad \boldsymbol{e}_t \leftarrow \mathrm{Decode}\big(\boldsymbol{h}_{t-1}^L\big)\,, \quad C = \boldsymbol{e}_0, \boldsymbol{e}_1, \ldots, \boldsymbol{e}_{t-1}. \tag{1}$$

Bandwidths of a model's horizontal axis vs. vertical axis. It is useful to separate the horizontal axis (across positions) from the vertical axis (across depth). *Horizontally*, dense attention is effectively full-bandwidth: when generating token t, the layer-ℓ state $\boldsymbol{h}_t^\ell$ can read the cached representations of every earlier position. *Vertically*, access is restricted: $\boldsymbol{h}_t^\ell$ cannot read any deeper past state $\boldsymbol{h}_{t'}^{\ell'}$ with $t' < t$ and $\ell' \geq \ell$ (Fig. 1, left). Formally, the states reachable when computing position t at layer ℓ are

$$\mathcal{R}_{\text{std}}(t,\ell) = \big\{(t',\ell') : t' < t,\ \ell' < \ell\big\}, \qquad \big|\mathcal{R}_{\text{std}}\big| = \Theta(t\ell), \tag{2}$$

so a shallow layer of a new token sees only a *partially processed* view of the past, even though the deeper, more fully processed states of those same positions have already been computed and sit in the KV cache. Past computation produced at layer ℓ is readable only to layers above ℓ and can never be routed back down for further processing. This is the narrow vertical channel Sec. 3.1 widens.

Importantly, this depth-wise dependency constraint is also what lets transformers train in parallel across positions: sequential computation is required only across layers, not across tokens. At decoding time, however, generation is already sequential over tokens, so the constraint buys nothing, opening the door to richer dependencies on past hidden states, which we develop next.

3 Widening the Bandwidth with Latent Feedback Decoding

3.1 Latent Feedback Decoding

The central technique in full-bandwidth transformers is latent feedback decoding, which feeds the previous top-layer hidden state back into the input. At step t,

$$\boldsymbol{h}_t^L = f_\theta\big(\boldsymbol{e}_t \otimes \boldsymbol{h}_{t-1}^L;\ C\big)\,, \text{ where } \boldsymbol{e}_t \leftarrow \text{Decode}\big(\boldsymbol{h}_{t-1}^L\big)\,,\ C = \boldsymbol{e}_1, \boldsymbol{e}_2 \otimes \boldsymbol{h}_0^1, \ldots, \boldsymbol{e}_t \otimes \boldsymbol{h}_{t-1}^L \tag{3}$$

where f_θ is the L-layer transformer stack, $\cdot \otimes \cdot$ fuses the sampled token's embedding with the previous latent state, and C is the past context (the KV cache of all earlier positions).

The fusion $\otimes$ is a gated linear unit:

$$\boldsymbol{e}_t \otimes \boldsymbol{h}_{t-1} = \boldsymbol{W}^H \boldsymbol{h}_{t-1} \odot \sigma(\boldsymbol{W}^G \boldsymbol{e}_t), \tag{4}$$

with $\boldsymbol{W}^H, \boldsymbol{W}^G \in \mathbb{R}^{D\times D}$. The asymmetry is deliberate: the hidden state occupies the value pathway, while the token embedding enters only as a multiplicative gate. We hypothesize that a symmetric fusion such as $\boldsymbol{W}^G \boldsymbol{e}_t + \boldsymbol{W}^H \boldsymbol{h}_{t-1}$ would leave a shortcut open where the model could ignore the $\boldsymbol{h}$ and recover the plain token input. That shortcut is tempting when training starts from a standard checkpoint where ignoring $\boldsymbol{h}$ could give lower loss. However, in small scale ablation study, we do not observe evidence of such shortcut under symmetric fusions (App. S.1).

Latent feedback is free to serve. The added inference cost is independent of context length and model depth. The state $\boldsymbol{h}_{t-1}^L$ is already computed during standard decoding, so the only extra work is two $D \times D$ matrix multiplications, negligible against a forward pass through L blocks ($< 1\%$). Since fusion preserves the input dimension D, the architecture, KV-cache layout, and serving stack are untouched, and the decoding loop changes by two lines (Fig. 2, right). The scheme is also vLLM-compatible: we store top-layer states in a dedicated buffer, adapting the mechanism used by multi-token-prediction implementations (Appendix E). The end-to-end latency results under vLLM are shown in Appendix P.

Latent feedback improves computational accessibility. The main benefit of latent feedback decoding comes from lifting the depth restriction of Eq. (2). Now every layer, including the lowest, can access top-layer information,

$$\mathcal{R}_{\text{lf}}(t,\ell) \ = \ \big\{\,(t',\ell')\ :\ t' < t,\ 0 \leq \ell' \leq L\,\big\}, \qquad \big|\mathcal{R}_{\text{lf}}\big| = \Theta(TL), \tag{5}$$

shown in Fig. 1 (right). The improved accessibility is empirically verified in synthetic settings in Sec. Q and leads to improved performance in free-form generation tasks (Sec. 4.2).

Latent feedback adds draft space. Latent feedback also provides a continuous scratchpad, relieving the pressure to verbalize intermediate state. State maintenance moves from the sequence axis alone to the depth axis as well: intermediate results can be updated recurrently over time, through $\boldsymbol{h}$ along the stack rather than only by extending the token sequence. This predicts the possibility of shorter rollouts, for which Sec. 4.3 provides preliminary evidence.

Listing 1: Training: one step with k passes.

```
def glu_cross(h, e): # [T,D],[T,D]->[T,D]
    return (h @ W_u) * sigmoid(e @ W_g)

e = embed(tokens)   # [T, D]
h = model(e)        # pass 1 (standard)
loss = ntp_loss(h)
for _ in range(k - 1): # parallel in T
    x = glu_cross(shift_right(h), e)
    # random plain prefix
    x = prefix_mixin(x, e)
    h = model(x)
    loss += ntp_loss(h) / (k-1)
```

Listing 2: Inference (SOFT); uncommenting line 2-3 gives FUSED; line 7 shows the STANDARD-decoding input).

```
h = model(embed(prompt)) # prefill, h: [T, D]
#h = model(
#   glu_cross(shift_right(h), embed(prompt)))
tok = sample(lm_head(h[-1]))
h_prev = h[-1]
while not done:              # decode
    x = glu_cross(h_prev, embed(tok))
    # standard decoding: x = embed(tok)
    h_prev = model_step(x, kv_cache)
    tok = sample(lm_head(h_prev))
```

Figure 2: **Latent feedback in pseudo-code (simplified).** Training (left) pays sequentiality across k passes, each parallel over positions. Inference (right) differs from standard decoding by a single line (line 6 vs. the commented line 7): the input is the fused state rather than the token embedding alone, reusing the state previously used for decoding. Fig. 7 in the appendix shows the full training and inference procedure with additional stability techniques such as input normalization included.

What latent feedback does NOT provide. We provide two important clarifications:

- **No mutable register.** RNNs and state-space models overwrite a compressed state at each step. Latent feedback is recurrent in form, but past states persist in the KV cache rather than being overwritten, so every earlier state remains readable by the current token.
- **No added asymptotic depth at decoding time.** Latent feedback does not change the asymptotic serial depth of decoding: with or without it, each step has a depth-$\mathcal{O}(L)$ graph, so T tokens cost $\mathcal{O}(TL)$. What changes is the *bandwidth* of the path, with a verbal channel and a continuous channel now *evolving in parallel*. Note that a full-bandwidth transformer can further increase the depth at prefilling time through a multipass prefill, which we will introduce in the following section.

3.2 PARALLEL MULTI-PASS TRAINING FOR LATENT FEEDBACK DECODING

At decoding time, latent feedback unrolls over generated positions. Let $\boldsymbol{u}_t$ be the input actually fed to the transformer stack at position t. The input sequence looks like

$$\boldsymbol{u}_1 = \boldsymbol{e}_1, \quad \boldsymbol{h}_1 = f_\theta(\boldsymbol{u}_1; C_1); \qquad \boldsymbol{u}_t = \boldsymbol{e}_t \otimes \boldsymbol{h}_{t-1}, \quad \boldsymbol{h}_t = f_\theta(\boldsymbol{u}_t; C_t), \quad t \geq 2. \tag{6}$$

Here $\cdot \otimes \cdot$ is the gated fusion of Eq. (4), and C_t is the KV cache over the previous inputs $\boldsymbol{u}_{1:t-1}$. Thus the stack sees the input sequence $\boldsymbol{e}_1,\ \boldsymbol{e}_2 \otimes \boldsymbol{h}_1,\ \boldsymbol{e}_3 \otimes \boldsymbol{h}_2,\ \boldsymbol{e}_4 \otimes \boldsymbol{h}_3, \ldots$ rather than plain embeddings alone. Since a standard next-token-prediction model is trained only on plain token embeddings in this slot, full-bandwidth transformers must be trained on these latent-feedback inputs as well.

The exact recurrence of Eq. (6) is sequential in t, so training on it directly requires RNN-style backpropagation through time (BPTT), which breaks the parallel teacher forcing. We instead adopt a multi-forward-pass approximation. For each position in the sequence, we compute the top-layer state several times, writing $\boldsymbol{h}_t^{(k)}$ for the state at position t on pass k (the layer superscript L is omitted throughout this section):

$$\boldsymbol{h}_t^{(1)} = f_\theta(\boldsymbol{e}_t;\, C^{(1)}), \qquad C^{(1)} = \boldsymbol{e}_1, \ldots, \boldsymbol{e}_{t-1}, \tag{7}$$

$$\boldsymbol{h}_t^{(2)} = f_\theta\big(\boldsymbol{e}_t \otimes \boldsymbol{h}_{t-1}^{(1)};\, C^{(2)}\big), \qquad C^{(2)} = \boldsymbol{e}_1,\, \boldsymbol{e}_2 \otimes \boldsymbol{h}_1^{(1)},\, \ldots,\, \boldsymbol{e}_{t-1} \otimes \boldsymbol{h}_{t-2}^{(1)}, \tag{8}$$

$$\cdots$$

$$\boldsymbol{h}_t^{(k)} = f_\theta\big(\boldsymbol{e}_t \otimes \boldsymbol{h}_{t-1}^{(k-1)};\, C^{(k)}\big), \qquad C^{(k)} = \boldsymbol{e}_1,\, \boldsymbol{e}_2 \otimes \boldsymbol{h}_1^{(k-1)},\, \ldots,\, \boldsymbol{e}_{t-1} \otimes \boldsymbol{h}_{t-2}^{(k-1)}. \tag{9}$$

The first pass is the ordinary no-feedback forward pass ($\boldsymbol{h}_t^{(1)} \equiv \boldsymbol{h}_t$); each subsequent pass shifts the previous pass's states one position rightward, fuses them with the token embeddings, and re-runs the full stack in parallel across all positions, since every state it requires was completed in the previous pass. We then apply the standard teacher-forced next-token-prediction loss to the outputs of *every pass*. Retaining the first-pass loss preserves the model's no-feedback mode of operation, which is what processes the prompt at inference time. We *do not detach the gradient*, so the loss from later

passes backpropagates into earlier passes' latent states, acting as an auxiliary objective; this does increase the memory footprint. The overall objective is

$$\mathcal{L}^K(\theta) = \underbrace{\sum_{t=1}^{T} -\log p_\theta(x_{t+1} \mid \boldsymbol{e}_{1:t})}_{\text{standard NTP objective}} + \lambda \frac{1}{K-1} \sum_{k=2}^{K} \sum_{t=1}^{T} -\log p_\theta\left(x_{t+1} \mid \boldsymbol{u}_{1:t}^{(k)}\right), \tag{10}$$

where $\boldsymbol{u}_{1:t}^{(k)} = \boldsymbol{e}_1, \boldsymbol{e}_2 \otimes \boldsymbol{h}_1^{(k-1)}, \ldots, \boldsymbol{e}_t \otimes \boldsymbol{h}_{t-1}^{(k-1)}$ are the pass-k fused inputs of Eqs. (8)–(9). In all experiments we set $\lambda = 1$ without any tuning.

The pseudocode and diagram is shown in Fig. 2 left and Fig. 8 in appendix. We refer to this scheme as *temporal parallelism*, following a common strategy for parallelizing recurrent computation during training (Zeng et al., 2025; Cai et al., 2026; Huang et al., 2026; Liu & Liu, 2026). After k passes, a top-layer state from position t can affect the input at positions up to $t + k - 1$, so k passes train the feedback transition over a horizon of $k - 1$ token steps. Training thus pays sequentiality across passes rather than across positions.

Feedback-pass scheduling. At decoding time, the feedback loop unrolls indefinitely, so the trained map must remain stable under many more self-compositions than any training budget can simulate; yet running many passes throughout training is prohibitively expensive, since each pass multiplies the cost of the run. Scheduling the number of forward passes—how many, and when—is therefore central to making latent-feedback training practical.

How many passes. We choose the number of passes based on *empirically* checking whether the iterated feedback map reaches a stable state. Our experiments show that full-bandwidth transformer takes only 3 forward passes to reach a stable state, where further forward passes do not change the hidden state or validation loss. We hypothesize there are two factors contributing to the fast convergence: (1). Each pass's input is only partially changed: The token embedding is fixed, and only the hidden-state is changing; (2). We apply NTP supervision on the output of each pass (analogous to intermediate states in loop transformer), a technique critical for stability and faster convergence of loop transformer suggested by recent works (Zhu et al., 2025; Sharma & Vu, 2026; Shomali et al., 2026).

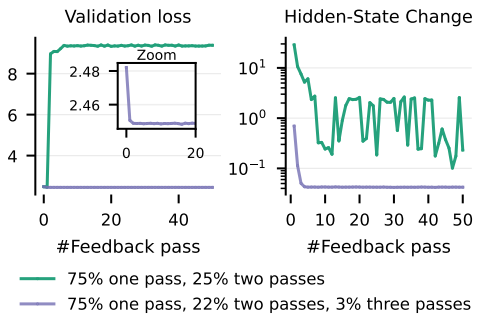


Figure 3: **Three-pass training stabilizes long-horizon latent feedback.** With only single- and two-pass batches, validation loss and hidden-state updates become unstable beyond the training horizon. Adding a small fraction of three-pass batches yields stable loss and bounded hidden-state updates ($\mathbb{E}_t\left[\|\boldsymbol{h}_t^{(k)} - \boldsymbol{h}_t^{(k-1)}\|\right]$).

When to introduce feedback passes. Because feedback passes are expensive, most of training uses the standard single-pass objective. We introduce latent feedback progressively in the middle of training: first with two-pass batches, and later with a small fraction of batches with more passes. This lets the run begin from an ordinary pretrained checkpoint, spend the bulk of its compute on standard teacher forcing, and pay the extra feedback-pass cost only mid-training, when it is needed to stabilize the feedback map.

Fig. 3 illustrates the feasibility of the scheduling. We studied a 1B model trained on 200B tokens. A model trained with only single- and two-pass batches (75% single-pass, 25% two-pass; green) performs well at the trained depth but fails to extrapolate. Adding only 3% three-pass batches (75% single-pass, 22% two-pass, 3% three-pass; blue) changes the behavior: validation loss remains flat through 30 feedback steps, and the hidden-state change decays to a small plateau. The same extrapolation behavior carries over to inference: hundred-token rollouts show no sign of breakdown (Fig. 5, solid green line), and we observe similar stability under $k = 1000$ feedback passes (Fig. 9c).

Lastly, during actual training, we included additional techniques, including randomly mixing in standard embeddings in the prefix feedback pass inputs to reduce the train-inference gap (prefix mixin), jitter noise for smoothing the hidden states, as well as extra input normalization and depth scaling for stable norm during recursion. The full description is in Appendix D.

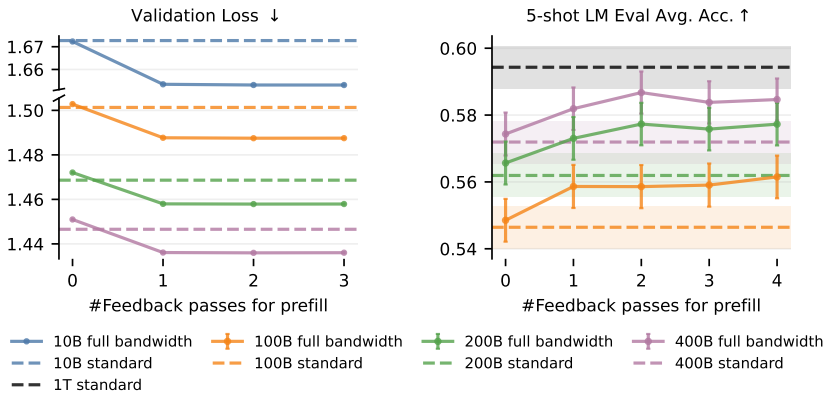


Figure 4: **Feedback passes during prefilling improve non-generative performance.** Under a 1B model trained under different number of tokens, re-running prefill with additional fused passes (Eqs. (8)–(9)) improves both validation perplexity (left) and 5-shot LM Eval accuracy across 10 tasks (right) across training scales; Error bars denote one standard error of the mean, obtained by propagating individual per-task standard errors (summing in quadrature and dividing by the number of tasks). The 400B token model refers to the $1.28\times$ extra compute version, the $1.5\times$ compute version shows stronger results, shown in Fig. 9d in appendix.

4 Experiments

To evaluate full-bandwidth transformers, we pretrain 1B-parameter models (architecture described in Appendix C) using the training recipe from Sec. 3.2. We use NorMuon (Li et al., 2026) for matrix parameters with learning rate 1×10^{-2} and weight decay 0.01, and Adam for all other parameters with learning rate 5×10^{-4} and no weight decay. All runs use a WSD learning-rate schedule (Hägele et al., 2024; Hu et al., 2024) with 200 warmup steps and a 25% cooldown phase decaying to zero. During cooldown, we add a z-loss (Chowdhery et al., 2023) with coefficient 1×10^{-5} and decay weight decay together with the learning rate following AdamC (Defazio, 2025). For all experiments, we use a jitter noise with $\sigma = 0.02$ (Eq. (11)) during training. Models are trained on the data mixture of Phi-4 (Abdin et al., 2024), with context length 8192. For all runs, we use a global batch size of 300K tokens; except for the 1T-token baseline run, which uses a larger global batch size of 1.2M tokens. We summarize below the exact scheduling and full-bandwidth transformer runs' *token-equivalent compute*, defined as training tokens multiplied by the average number of forward passes per batch.

Run	Feedback-pass schedule	Tokens	Token-equivalent compute
10B	100% three-pass	10B	30B
100B	75% one-pass, 25% three-pass	100B	150B
200B	75% one-pass, 22% two-pass, 3% three-pass	200B	256B
400B ($1.28\times$)	75% one-pass, 22% two-pass, 3% three-pass	400B	512B
400B ($1.5\times$)	75% one-pass, 25% three-pass	400B	600B

Example validation loss dynamics under 400B tokens run, of different passes at different schedules, are shown in Fig. 17 in the appendix. The actual per-step training time under different number of passes and the inference latency are shown in App. G and App. P.

4.1 Fused prefilling improves non-generative performance

Latent-feedback training enables a simple form of prefill-time test-time scaling. At evaluation, we apply k additional fused passes over the prompt using Eqs. (8)–(9). These passes augment embeddings with richer context information from top-layer hidden states, improving perplexity and downstream accuracy.

Fig. 4 plots validation loss and average 5-shot LM Eval accuracy across RTE, TruthfulQA-MC2, ARC-Easy, ARC-Challenge, BoolQ, PIQA, WinoGrande, OpenBookQA, COPA, and MMLU, as a function of the number of feedback passes applied during prefill. Step 0 is ordinary prefill using embeddings as input. Step k corresponds to k extra forward passes. Broadly, most of the improvement

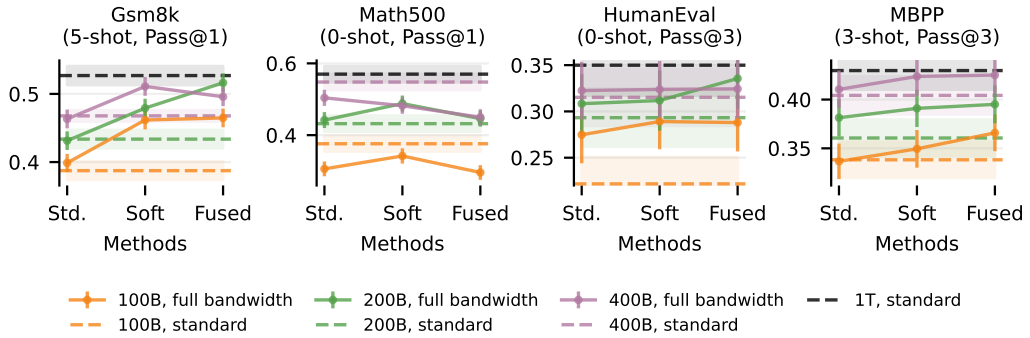


Figure 5: **Latent-feedback decoding improves pre-trained base 1B parameter models at fixed decoding cost.** We compare the three decoding regimes from Sec. 4.2. Coding results report Pass@3 from 10 rollouts, with temperature selected from $\{0.3, 0.5, 0.7\}$ separately for each method (for *both* baseline and our model). SOFT generally improves over STANDARD at identical inference cost, while FUSED can yield further gains with $2\times$ prefill compute and unchanged decoding cost. The MATH-500 degradation observed before instruction tuning is reversed after instruction tuning, with the best full-bandwidth regime matching or exceeding the standard baseline at each scale (Table 1).

appears after the top-layer hidden states are first made available at the input. Further passes continue to help, but with diminishing returns. In addition, *latent feedback training costs little when unused*. At step 0, where the model is evaluated as an ordinary transformer, full-bandwidth transformers give up only a small amount of validation loss relative to the standard baseline, while already improving average LM Eval accuracy. Notably, a small amount of prefill-time compute matches substantially larger standard baselines: With two feedback passes, the 100B-token full-bandwidth transformer reaches the 200B-token standard baseline, and the 200B-token full-bandwidth transformer reaches the 400B-token standard baseline. Lastly, the 400B full bandwidth model refers to the $1.28\times$ version, the $1.5\times$ compute version (Fig. 9d in appendix) gives stronger results under more training time budget.

We additionally compare our model with other models of similar parameter scale on 0-shot LM Eval performance, shown in Table 2 in Appendix I, where we found our model performs on-par or better than models trained under similar or more tokens. These results imply that full-bandwidth transformers improve on a strong baseline.

4.2 LATENT FEEDBACK DECODING IMPROVES DECODING PERFORMANCE

We now evaluate open-ended generation tasks. We compare three decoding regimes:

- STANDARD: single-pass prefill; generation uses token embeddings only. This evaluates the full-bandwidth model as an ordinary transformer, and measures the cost of latent-feedback training when the feedback channel is not used at inference.
- SOFT: single-pass prefill; generation uses latent feedback as in Eq. (3). Same per-token overhead as except for the two $D \times D$ matrix multiplications.
- FUSED: the prompt is first processed by an additional fused prefill pass for refinement, as in Eq. (8); generation then proceeds as in SOFT.

Thus STANDARD and SOFT have identical prefill cost, while FUSED doubles prefill cost while keeping the same per-token decoding cost as SOFT and STANDARD.

Evaluation setting. We evaluate on GSM8K (Cobbe et al., 2021), MATH-500 (Lightman et al., 2023), HumanEval (Chen et al., 2021), and MBPP (Austin et al., 2021). We report Pass@1 for math and Pass@3 for coding. For coding, Pass@3 is estimated from 10 rollouts per problem, with temperature grid-searched over $\{0.3, 0.5, 0.7\}$ separately for each decoding regime. We do not use top-k or top-p. The median generation length for each task is shown in Table 4 in appendix.

Latent feedback decoding improves the base model. Fig. 5 compares the three decoding regimes for full-bandwidth models (solid lines) against standard baselines (dashed lines). Similar to Fig. I, the 400B full bandwidth model refers to the $1.28\times$ version and the $1.5\times$ shows consistently better

	FBT, 200B			FBT, 400B (1.28×)			FBT, 400B (1.5×)			Standard transformer		
Task	STD.	SOFT	FUSED	STD.	SOFT	FUSED	STD.	SOFT	FUSED	200B	400B	1T
GSM8K	64.52	**67.93**	67.55	67.90	71.00	**71.80**	68.08	70.74	**73.09**	62.93	68.39	70.13
MATH-500	43.80	**45.60**	**45.60**	46.00	45.40	**48.40**	45.40	**49.00**	48.80	42.40	46.40	47.40
HumanEval	42.54	45.06	**45.92**	46.50	47.20	**47.60**	45.69	46.35	**48.77**	37.16	44.85	50.01
MBPP	38.39	39.80	**41.22**	40.50	40.60	**41.70**	40.23	41.36	**42.34**	38.61	40.28	41.93

Table 1: **Latent-feedback gains carry over through instruction tuning.** We evaluate full-bandwidth transformers after long-context extension and instruction tuning, using no few-shot examples. Scores are percentages. For math tasks, we report Pass@1; for coding tasks, we report Pass@3 estimated from 10 rollouts per problem, selecting the best temperature from $\{0.3, 0.5, 0.7\}$ for each setting. Bold indicates the best decoding regime within each full-bandwidth training scale. Results on instruction following and creative writing benchmarks are presented in table 3 in the appendix, where we do not find the performance varying significantly with decoding regime.

performance (red lines, Fig. 9e). Across GSM8K and coding tasks, SOFT generally improves over STANDARD at the same decoding cost, while FUSED can provide further gains with 2× prefill compute. MATH-500 is the main exception: before instruction tuning, latent feedback does not consistently improve over STANDARD, particularly at larger scale. On the remaining tasks, full bandwidth models often approach or exceed standard baselines trained with 2–5× more tokens, including the 1T baseline on GSM8K and HumanEval. Finally, Pass@3 improvement on coding tasks suggests that conditioning generation on recurrent hidden states does not reduce sampling diversity.

The improvement carries over through instruction tuning. We further apply long-context extension (12B tokens) from 8K to 32K and instruction tuning (6B tokens) for models of different pre-train budgets and then evaluate without few-shot examples. Because these stages are much shorter than pretraining, we train them with *three* forward passes throughout rather than using the pretraining feedback-pass schedule[1]. Results are shown in Table 1, where the gains from latent feedback largely persist after instruction tuning. SOFT and FUSED are generally competitive with or improve upon STANDARD, and the best latent-feedback regime outperforms the matched-token standard baseline on all four tasks at every reported scale. In particular, the MATH-500 degradation observed in the base models disappears after instruction tuning. At 400B (1.5×), SOFT also exceeds the 1T-token standard baseline on MATH-500 (49.0 vs. 47.4). Table 3 reports additional instruction-following and writing benchmarks, where we observe little difference between approaches. Finally, increasing the number of feedback passes for FUSED does not yield further gains (Fig. 11).

4.3 LATENT FEEDBACK ENABLES MORE CONCISE REASONING

On the base model and 0-shot setting, SOFT decoding often produces shorter reasoning traces than STANDARD Fig. 6 shows the median rollout length on a 200B token model (Fig. 10 in appendix observe same patterns observed in other runs), Fig. 20 in Appendix T shows example shortened rollouts. This is consistent with the behavior predicted by the widened feedback channel: Intermediate computation that would otherwise need to be represented through generated tokens may instead be carried in the recurrent hidden state, reducing the nu

Notably, the effect is less pronounced after instruction tuning or given few shot examples. We hypothesize the cause as external reasoning examples being off-policy with respect to latent-feedback decoding: the target traces were produced by standard token-by-token reasoning, so fitting them re-imposes the fully verbalized style regardless of what the state can carry. On-policy post-training (Ma et al., 2026) under latent feedback may preserve the conciseness, which we leave to future work.

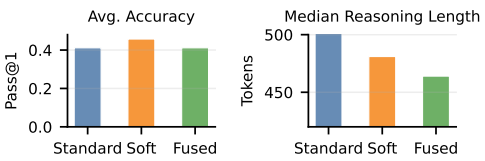


Figure 6: Reasoning length and accuracy on Math500 from the 200B run (green line, Fig. 5).

[1]Baseline standard models are trained on the same set and number of tokens since there is no extra corpus available for spending more compute on.

4.4 SYNTHETIC DATA EXPERIMENTS AND ABLATION STUDY

State extraction and tracking on synthetic data In appendix Q, we generate synthetic sequences and probe the ending token's residual about states determined by tokens at the beginning of the sentence. When prefilled with latent feedback at last few tokens, shallow layers show significantly higher accuracy than under standard prefilling in shallow layers, since latent feedback exposes deep layer states, which include global information, to shallow ones.

Ablation study In appendix S, we inspect the learned gating weights, $\boldsymbol{W}^H, \boldsymbol{W}^G$ from Eq. (4), where we observe both pathways contributing non-trivially. We confirm the empirical performance benefit of *not* detaching hidden state at each pass and the role of injected jitter noise (Eq. (11)). We also tested alternate scheduling where multi-forward pass step interleave with one-pass step, which prevents continuing from a mid-train checkpoint and shows suboptimal performance compared with having each stage in bulk.

5 RELATED WORK, LIMITATIONS AND FUTURE WORK

We provide more extensive related work discussion in Appendix B, which covers continuous reasoning, parallel training of recurrent networks and data efficient pre-training.

Latent recurrence across decoding steps. One central idea behind our work is to overlap recurrent compute with the sequential decoding process. There are other works considering similar ideas. *Feedback Transformer* (Fan et al., 2020) is the pioneering work along this line; At each position, they generate a mixture of each layer's representation and let attention in future positions attend to the mixture rather than the same-layer key values. However, their training is sequential over input tokens, limiting their scalability, whereas our training is parallelized over all positions. Many concurrent works explore a similar direction. T^2*MLR* (Cai et al., 2026), *Latent Recurrent Transformer* (Huang et al., 2026), *Recirculation* (Mozer et al., 2026) consider recurrence with middle-layer injection, where the previous position's hidden state at a late middle layer is fused with the current position's early middle layer hidden state. Methodology-wise, our approach is similar in the training approach and the motivation. Our approach mainly differs in the point of reinjection; specifically, our injection happens "externally" to the model and therefore introduces no architecture changes. We also introduce the least amount of extra parameters. For a L-layer transformer with D-dimension residual, we introduce only two linear projections (each of size $D \times D$), in contrast to T^2MLR's extra MLPs ($5D^2$ parameters) and LRT's layerwise projection, which introduces LD^2 parameters. The bigger difference lies in the scale of empirical evaluation: our work performs the largest-scale pre-training (up to 400B tokens) among all works, enabled by feedback pass scheduling; therefore, we manage to verify the actual *decoding time improvement* on free-form generation tasks, whereas LRT only considers non-freeform eval (similar to our setting in Fig. 4 right), and T^2MLR considers synthetic state tracking tasks and GSM8K only after fine-tuning the model on the math corpus. However, considering the similarity in spirit, we do not foresee reasons why one approach would outperform the others, and exactly which method (and more broadly, which form of past hidden state injection) works the best *at large scale* remains unclear since we do not have the resources for verification. Very recent work also explores latent feedback under prefill–decode decoupled architectures, including Maglev (Liu & Liu, 2026) and Recurrent Looped Transformer (Zhang, 2026), which pairs a context encoder with a recurrent decoder that feeds its output hidden state into the next token's input.

Loop transformers. Our approach is similar to loop transformers (Fan et al., 2026; Dehghani et al., 2018; Giannou et al., 2023; Geiping et al., 2025) during training in that the model's outputs are repeatedly fed back as inputs across multiple forward passes. Differences lie in inference time: loop transformers are recurrent over depth, thereby increasing inference compute with the number of reapplying the transformer stack. In contrast, full-bandwidth transformer performs recurrence over time, integrated into the autoregressive decoding loop, which *avoids additional transformer-block evaluations* per generated token (with additional compute used only when optional multi-pass prefilling is used).

Limitations and future work. Due to space constraints, we present current work's limitations, potential solutions, and future work in App. A, including experiment scale, principled feedback pass scheduling, multi-token prediction compatibility, post-training recipe, interaction with efficient attention and CoT monitoring challenge.

AI USE STATEMENT

In this work, we used generative AI tools for paper polishing and experiment implementation. We have not used generative AI tools for proposing hypotheses and refining research ideas. We have reviewed all AI-assisted work, where proof read the paper manually. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The main results of the paper utilize internal, non-public dataset and training infrastructure. In the supplementary material, we provided an open-source version of the codebase based on NanoChat (https://github.com/karpathy/nanochat/), which gives similar observations, but not identical results.

REFERENCES

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Bowen Baker, Joost Huizinga, Leo Gao, Zehao Dou, Melody Y Guan, Aleksander Madry, Wojciech Zaremba, Jakub Pachocki, and David Farhi. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation. *arXiv preprint arXiv:2503.11926*, 2025.

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

Jonah Brown-Cohen, David Lindner, and Rohin Shah. Quantifying the necessity of chain of thought through opaque serial depth. *arXiv preprint arXiv:2603.09786*, 2026.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of machine learning research*, 24(240):1–113, 2023.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Beiya Dai, Yuliang Liu, Daozheng Xue, Yunchong Song, Qipeng Guo, Kai Chen, Xinbing Wang, Bowen Zhou, and Zhouhan Lin. Context-level language modeling by learning predictive context embeddings. *arXiv preprint arXiv:2510.20280*, 2025.

Federico Danieli, Pau Rodriguez, Miguel Sarabia, Xavier Suau, and Luca Zappella. Pararnn: Unlocking parallel training of nonlinear rnns for large language models. *arXiv preprint arXiv:2510.21450*, 2025.

Aaron Defazio. Why gradients rapidly increase near the end of training. *arXiv preprint arXiv:2506.02285*, 2025.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Joshua Engels, Callum McDougall, Bilal Chughtai, Janos Kramar, Senthoran Rajamanoharan, Cindy Wu, Arthur Conmy, Asic Q Chen, Jean Tarbouriech, Min Ma, et al. How transparent is diffusiongemma? *arXiv preprint arXiv:2606.20560*, 2026.

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.

Ying Fan, Anej Svete, and Kangwook Lee. Bridging the gap between latent and explicit reasoning with looped transformers. *arXiv preprint arXiv:2606.31779*, 2026.

Jonas Geiping, Sean Michael 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. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=S3GhJooWIC.

Angeliki Giannou, Shashank Rajput, Jy-yong Sohn, Kangwook Lee, Jason D Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning*, pp. 11398–11442. PMLR, 2023.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. In *First conference on language modeling*, 2024.

Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna B Allal, Leandro Von Werra, and Martin Jaggi. Scaling laws and compute-optimal training beyond fixed training durations. *Advances in Neural Information Processing Systems*, 37:76232–76264, 2024.

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.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

Zeyi Huang, Xuehai He, LiLiang Ren, Yiping Wang, Baolin Peng, Hao Cheng, Shuohang Wang, Pengcheng He, Jianfeng Gao, Yong Jae Lee, et al. Latent recurrent transformer: Architecture exploration, training strategies, and scaling behavior. *arXiv preprint arXiv:2605.26797*, 2026.

Albert Q Jiang, A Sablayrolles, A Mensch, C Bamford, D Singh Chaplot, Ddl Casas, F Bressand, G Lengyel, G Lample, L Saulnier, et al. Mistral 7b. arxiv. *arXiv preprint arXiv:2310.06825*, 10(3), 2023.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024a.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to solve inherently serial problems. In *International Conference on Learning Representations*, volume 2024, pp. 11911–11943, 2024b.

Zichong Li, Liming Liu, Chen Liang, Weizhu Chen, and Tuo Zhao. Normuon: Making muon more efficient and scalable. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=m1IRWFAMsa.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Bo Liu and Qiang Liu. Maglev: Sliding recurrent memory. *arXiv preprint arXiv:2608.02870*, 2026.

Yuliang Liu, Yunchong Song, Yixuan Wang, Kewen Ge, Alex Lamb, Qipeng Guo, Kai Chen, Bowen Zhou, and Zhouhan Lin. Next concept prediction in discrete latent space leads to stronger language models. *arXiv preprint arXiv:2602.08984*, 2026.

Wenhan Ma, Hailin Zhang, Liang Zhao, Yifan Song, Yudong Wang, Zhifang Sui, and Fuli Luo. Stabilizing moe reinforcement learning by aligning training and inference routers. *arXiv preprint arXiv:2510.11370*, 2025.

Wenhan Ma, Jianyu Wei, Liang Zhao, Hailin Zhang, Bangjun Xiao, Lei Li, Qibin Yang, Bofei Gao, Yudong Wang, Rang Li, et al. Mopd: Multi-teacher on-policy distillation for capability integration in llm post-training. *arXiv preprint arXiv:2606.30406*, 2026.

Bishwas Mandal, Shmuel Berman, Akshay Vegesna, and Samip Dahal. q0: Primitives for hyper-epoch pretraining. *arXiv preprint arXiv:2606.03938*, 2026.

Sajad Movahedi, Vera Milovanović, Shlomo Libo Feigin, Alexander Theus, Thomas Hofmann, Valentina Boeva, T Konstantin Rusch, and Antonio Orvieto. Fixed-point reasoners: Stable and adaptive deep looped transformers. *arXiv preprint arXiv:2606.18206*, 2026.

Michael C Mozer, Shoaib Ahmed Siddiqui, Danny Sawyer, Sunny Sanyal, and Rosanne Liu. Recirculation. *arXiv preprint arXiv:2608.17981*, 2026.

Lorenzo Noci, Sotiris Anagnostidis, Luca Biggio, Antonio Orvieto, Sidak Pal Singh, and Aurelien Lucchi. Signal propagation in transformers: Theoretical perspectives and the role of rank collapse. *Advances in Neural Information Processing Systems*, 35:27198–27211, 2022.

Zhenting Qi, Fan Nie, Alexandre Alahi, James Zou, Himabindu Lakkaraju, Yilun Du, Eric Xing, Sham Kakade, and Hanlin Zhang. Evolm: In search of lost language model training dynamics. *arXiv preprint arXiv:2506.16029*, 2025.

Rituraj Sharma and Tu Vu. Dense supervision is not enough: The readout blind spot in looped language models. *arXiv preprint arXiv:2606.24898*, 2026.

Behzad Shomali, Markus Frey, David Berghaus, Joachim Koehler, and Mehdi Ali. Loopmtp: A looped transformer guided by latent multi-token prediction. *arXiv preprint arXiv:2608.03624*, 2026.

Varun Singh, Lucas Krauss, Sami Jaghouar, Matej Sirovatka, Charles Goddard, Fares Obied, Jack Min Ong, Jannik Straube, Aria Harley, Conner Stewart, et al. Arcee trinity large technical report. *arXiv preprint arXiv:2602.17004*, 2026.

Jayden Teoh, Manan Tomar, Kwangjun Ahn, Edward S Hu, Tim Pearce, Pratyusha Sharma, Akshay Krishnamurthy, Riashat Islam, Alex Lamb, and John Langford. Next-latent prediction transformers learn compact world models. *arXiv preprint arXiv:2511.05963*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Yuning Wu, Jiahao Mei, Ming Yan, Chenliang Li, Shaopeng Lai, Yuran Ren, Zijia Wang, Ji Zhang, Mengyue Wu, Qin Jin, et al. Writingbench: A comprehensive benchmark for generative writing. *Advances in Neural Information Processing Systems*, 38, 2026.

Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. Tensor programs VI: Feature learning in infinite depth neural networks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=17pVDnpwwl.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *International Conference on Learning Representations*, volume 2025, pp. 29687–29707, 2025.

Zhenrui Yue, Bowen Jin, Huimin Zeng, Honglei Zhuang, Zhen Qin, Jinsung Yoon, Lanyu Shang, Jiawei Han, and Dong Wang. Hybrid latent reasoning via reinforcement learning. *Advances in Neural Information Processing Systems*, 38:5501–5530, 2026.

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

Xiangdong Zhang, Debing Zhang, Shaofeng Zhang, Xiaohan Qin, Yu Cheng, and Junchi Yan. Nitp: Next implicit token prediction for llm pre-training. In *Forty-third International Conference on Machine Learning*, 2026a.

Yifan Zhang. Recurrent looped transformer. Technical report, September 2026. URL https://github.com/yifanzhang-pro/recurrent-looped-tranformer.

Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, and Xin Wang. Soft thinking: Unlocking the reasoning potential of llms in continuous concept space. *Advances in Neural Information Processing Systems*, 38:168990–169012, 2026b.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

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

A LIMITATIONS AND FUTURE WORK

Larger scale experiments Our experiment scale is limited to 1B parameter models, and we did not verify the approach on models of larger scale. However, we believe latent feedback decoding can potentially introduce more benefit for larger ones with more number of layers where the top layer hidden state includes even richer information. More importantly, the ability for shallow layer to read deep layer information as well as the ability to recursively update a continuous state, is *not* something scaling model size or training horizon can acquire.

More principled scheduling Current feedback pass scheduling is based on a heuristic; future work can consider more rigorous ablation on the length of the recurrence training phase as well as a more principled approach to determine the number of recurrence steps, e.g., via the Jacobi iteration convergence diagnostics from Zeng et al. (2025).

Compatibility with multi-token prediction Multi-token prediction (Gloeckle et al., 2024, MTP) generates more than one token at each step at decoding time. However, latent feedback decoding requires a token's previous hidden state to generate the next token; therefore, the current form is incompatible with MTP. We believe future work can consider further incorporating next latent prediction objective (Teoh et al., 2025) during pre-training, which enables the model to propose multiple future hidden states and use it to predict multiple future tokens with latent feedback decoding.

Turning pre-trained model into full-bandwidth transformer Our feedback-pass scheduling (Sec. 3.2) effectively turns the full-bandwidth transformer's training into mid-training style, where the majority of the training stays identical as standard pre-training. Future work can further explore the feasibility of turning publicly available pre-cooldown models, such as the checkpoints from Arcee (Singh et al., 2026)[2] or just pre-trained base models into full-bandwidth transformers.

Post-training Certain architecture changes require modifications of the post-training algorithm. Particularly, the likelihood evaluation of rollouts under latent feedback now requires a recursive process over all positions for the policy model. This is fine for on-policy setting, where the hidden states and the likelihood can be captured during decoding. However, this can be tricky in off-policy setting, which involves evaluating the likelihood on the policy model position by position. However, one can consider utilizing multi-forward passes with shifted inputs as a multi-gram approximation to the true likelihood. We verify the feasibility in Fig. 12 in the appendix N.

Interaction with sliding window attention Our approach has some particular benefit for sliding window attention (SWA, Beltagy et al., 2020; Jiang et al., 2023). In particular, SWA is known to have small receptive fields. However, full-bandwidth transformers short-circuit the top-layer hidden state directly to the bottom, which exposes global information to the shallow layers. Future work can consider ablating the backbone model's architecture and see whether full-bandwidth transformer enables a higher ratio of SWA for better inference time efficiency and smaller KV cache size. We provide a more detailed discussion in Appendix O.

Increased challenge for CoT monitoring Lastly, latent feedback decoding increased the steps of unverbalized compute between two CoT (Brown-Cohen et al., 2026; Engels et al., 2026) steps, therefore increasing the difficulty of CoT monitoring (Baker et al., 2025). Future work can consider building probing based monitoring tools that directly reads the hidden states rather than verbal CoT. The issue is also recently discussed in the community [3].

B EXTENDED RELATED WORK

Latent and continuous reasoning. Our approach feeds top layer latent into the context, similar to the central idea of latent reasoning approaches such as Coconut (Hao et al., 2024) and Soft Thinking (Zhang et al., 2026b). The biggest differences are : (a) We focus on pre-training; (b) We

[2] https://huggingface.co/arcee-ai/Trinity-Nano-Base-Pre-Anneal

[3] https://www.redwoodresearch.org/blog/an-operationalization-of-opaque-serial-depth

use the hidden state to "augment" the generation rather than replacing the discrete tokens, therefore our approach is easier to supervise (but we may be less token efficient). Hybrid Latent Reasoning via Reinforcement Learning (Yue et al., 2026) proposes to use both the top layer hidden state and the generated tokens' embedding at post-training time during rollout, however they did not utilize top layer hidden state but instead they use it to generate a weighted mixture of vocabulary embedding so it is unclear whether it improves the reachability as the full bandwidth transformer does. There are also works studying latent reasoning at pre-training time, in particular, PonderLM-2 (Zeng et al., 2025) considers an interleaved embedding / hidden state as the input. Notably, their training approach is similar to us in that they use multiple forward passes to replace sequential rollout, however their approach doubles the input length (as well as KV cache size) so they introduce more training and inference overhead than the full bandwidth transformer.

Parallel training of recurrent networks. Another related direction is parallel training of recurrent networks. Most applications of this consider the linear special case like Mamba (Gu & Dao, 2024) or Gated Deltanet (Yang et al., 2025). These are clearly powerful techniques with use in various architectures yet in all such uses they are hybridized with standard transformer layers which can compensate for the missing representational capacity inherited from the linear constraint. ParaRNN (Danieli et al., 2025) goes further by parallelizing training of nonlinear recurrent neural networks via decoupling the optimizations at each point in the process and using Newton's iterations to achieve convergence with results comparable to transformers for language modeling. This approach here goes the other way, constructing recurrence on transformers with results that improve over baseline transformers, and it appears that the approach here is significantly more efficient.

Data-efficient pre-training. Our work falls into the broad category of improving LLM pre-training's data efficiency, i.e., given the same model size and fixed data, how can we use more flops to build a more powerful model under fixed or more inference overhead. Existing approaches consider additional objectives (beyond NTP) on the representation (Liu et al., 2026; Zhang et al., 2026a; Dai et al., 2025; Teoh et al., 2025) that encourage the hidden state to contain richer information. There has also been a recent NanoGPT slow run competition[4] that studies this setting, where the official solution (Mandal et al., 2026) trains a deep ensemble of LLMs and distills them into a single one for better performance. Compared with these approaches, our framework uses additional training flops for unlocking a new type of decoding regime that gives a free performance boost at inference time. Additionally, we believe techniques can flow between literature, for example, the depth scaling we used has also been shown to be important for the stability of training loop transformers (Movahedi et al., 2026). Our empirical verification of recurrence scheduling also suggests the feasibility of introducing computationally intensive auxiliary objectives only later on in the training.

More broadly, these methods point to a shift in the relevant scaling axes for pre-training. Conventional scaling primarily varies model parameters and training tokens. However, in large-scale training, the feasible design space is also constrained by pod size of GPUs, wall-clock budget, and the availability of high-quality unique tokens. Once the token-per-parameter ratio and the accessible pool of high-quality data become binding, simply increasing the number of unique training tokens is no longer the only, or even the most direct, path to improvement. A promising axis is to spend more computation per unique token through recurrent, iterative, or feedback-based mechanisms.

C Model Architecture

The model is a decoder-only causal language model with a tied 100,352-token embedding and output head, 24 transformer layers, a 1,536-dimensional hidden state, and 6,656-dimensional SiLU GLU feed-forward blocks. Its gated grouped-query attention uses 16 query heads, 8 shared key/value heads, headwise gates, QK RMS normalization, and rotary positions over an 8,192-token context; most layers use a 2,048-token sliding window, while every sixth layer uses full attention. RMS normalization is applied around each residual block and at the final output.

[4] https://qlabs.sh/slowrun/

D COMPLETE TRAINING STABILITY RECIPE

Prefix mixin. A distribution mismatch remains between multi-pass training and inference. At decoding time a sequence is heterogeneous: prompt positions carry plain token embeddings (processed by a single prefill pass), while generated positions carry fused inputs. In the passes of Eqs. (8)–(9), by contrast, *every* position beyond the first is fused. A model trained only on fully-fused passes therefore encounters an out-of-distribution boundary at inference, precisely where the prompt ends and generation begins. To close this gap we apply a *prefix mixin*: in each pass beyond the first, we sample a random prefix length p and revert positions $t \leq p$ to plain embeddings, fusing only the suffix. Training thus covers sequences that switch from plain to fused inputs at an arbitrary point, i.e. the structure of single-prefill inference. Alternatively, the prompt itself can be run through a second, fused prefill pass so that all positions match the fused distribution; the mixin removes the need for this, but we support both, corresponding to the "identical or doubled prefill" overhead stated in the abstract.

Stability recipes for long feedback horizons. At inference time, latent feedback may be applied for hundreds or thousands of generated tokens, far beyond the few feedback passes used during training. We therefore use several lightweight stabilization techniques to keep the feedback map well behaved under long self-composition.

- **Stationary hidden-state scale.** We keep the magnitude of carried state $\boldsymbol{h}_t^L$ bounded as feedback is repeatedly applied. To prevent the top-layer state norm from growing with depth, we use depth scaling (Yang et al., 2024; Noci et al., 2022) so that $\left\|\boldsymbol{h}_t^L\right\| \sim \mathcal{O}(1)$ rather than $\mathcal{O}(L)$, as can occur in a standard pre-norm model. We also apply RMSNorm to the fused input $\boldsymbol{e}_t \otimes \boldsymbol{h}_{t-1}^L$ before feeding it into the model.
- **Shared input basis with weight tying.** The model processes two types of inputs: plain token embeddings during standard prefill, and fused hidden-state/token inputs during latent-feedback decoding. We therefore encourage the embedding space and top-layer hidden-state space to remain in a compatible basis by tying the weights of the embedding layer and readout layer, reducing the burden on the fusion weights to learn a large corrective rotation between the two input distributions.
- **Noise regularization.** During training, we add small jitter noise to the carried hidden state before fusion,
$$\boldsymbol{h}_t^L = f_\theta\left(\boldsymbol{e}_t \otimes \boldsymbol{h}_{t-1}^L + \epsilon; C\right), \qquad \epsilon \sim \mathrm{Uniform}[-\sigma, \sigma]^D. \tag{11}$$
This exposes the feedback map to a local neighborhood around each training state, making it less sensitive to small deviations that can accumulate over long feedback horizons.

The complete pseudo code for training where the tricks are adopted is presented in Fig. 7 in the appendix.

E VLLM COMPATIBILITY

The implementation on vLLM follows the same design pattern as EAGLE Li et al. (2024a) / MTP Gloeckle et al. (2024): it retains each request's latest trunk hidden state and copies it in place into a persistent, fixed-address model buffer before the next decode step, allowing CUDA graphs to capture the glu cross gate (Eq. (4)) inside `forward`. A patched `GPUModelRunner._model_forward` stores detached hidden states in a dictionary keyed by request ID, uses `query_start_loc` to map packed rows to requests, and removes completed requests. Our forward function then fuses the saved state with the next token embedding through the learned glu cross gate, then recycles the resulting hidden state. Unlike EAGLE/MTP, which send target hidden states to a separate speculative draft model, our model feeds its own state back into the same model to define the actual next-token distribution.

F FULL PSEUDO CODE FOR TRAINING AND INFERENCE

Listing 3: Training: one step with k passes.

```
def glu_cross(h, e):       # [T,D],[T,D]->[T,D]
    return (h @ W_u) * sigmoid(e @ W_g)

e = embed(tokens)          # [T, D]
h = model(e)               # pass 1 (standard)
loss = ntp_loss(h)
for _ in range(k - 1):     # parallel in T
    h = h + uniform(-delta, delta) # jitter noise
    x = glu_cross(shift_right(h), input_rmsnorm_1(e))
    x = prefix_mixin(x, e) # random plain prefix
    h = model(input_rmsnorm_2(x))
    loss += ntp_loss(h) / (k-1)
```

Listing 4: Inference (SOFT); uncommenting line 2-3 gives FUSED; line 7 shows the STANDARD-decoding input).

```
h = model(embed(prompt)) # prefill, h: [T, D]
#h = model(
#    glu_cross(shift_right(h), embed(prompt)))
tok = sample(lm_head(h[-1]))
h_prev = h[-1]
while not done:              # decode
    x = glu_cross(h_prev, input_rmsnorm_1(embed(tok)))
    # standard decoding: x = embed(tok)
    h_prev = model_step(input_rmsnorm_2(x), kv_cache)
    tok = sample(lm_head(h_prev))
```

Figure 7: Full training and inference code for full-bandwidth transformer, with normalization layer and regularization noise included. The jitter noise is not applied at inference time but the two rmsnorm still do.

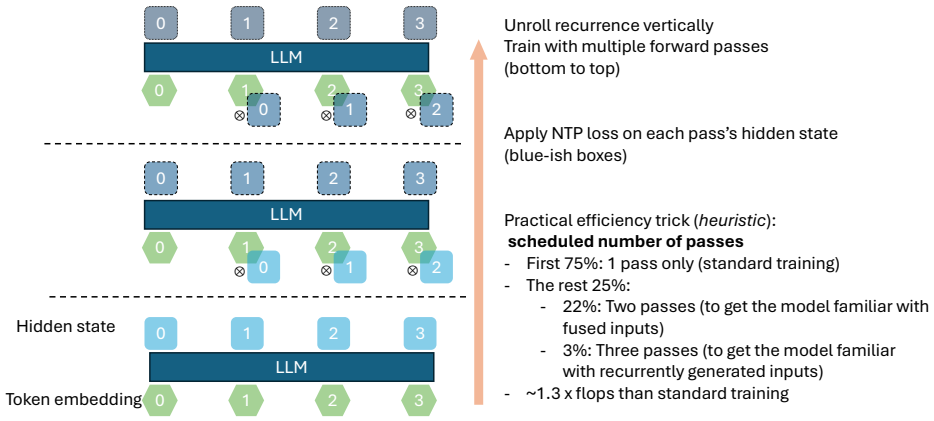


Figure 8: Illustration of multi feedback pass training

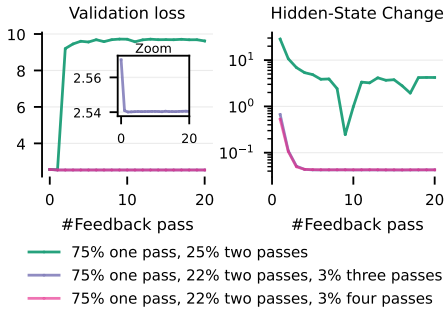
G END-TO-END PER STEP TRAINING TIME

Independent runs tested on 8 B200 GPUs in internal training infrastructure, with gradient checkpointing used to avoid OOM. Each additional pass adds approximately 0.4 seconds extra per iteration.

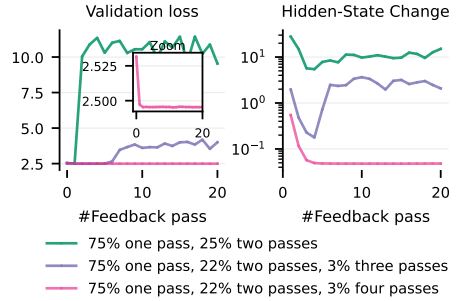
- 1 pass (standard ntp): 0.36 seconds
- 2 pass: 0.75 seconds
- 3 pass: 1.13 seconds
- 4 pass: 1.53 seconds

H FEEDBACK PASS SCHEDULING ABLATION

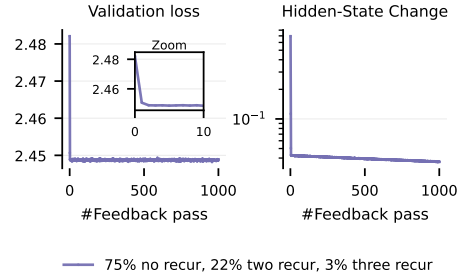
The schedule of feedback passes is based on heuristics, and pursuing extreme efficiency can result in instability. For instance, the 75% one pass, 22% two passes and 3% three passes schedule we suggested in the main text enables a stationary point when trained on 200B tokens (blue line, Fig.9a) but failed to extrapolate at 400B tokens scale (blue line, Fig. 9b and orange line, Fig. 9d). However changing the last stage from three to four passes or having three passes across the last 25% fixes the instability (purple line, Fig. 9b; blue and red lines, Fig. 9d).



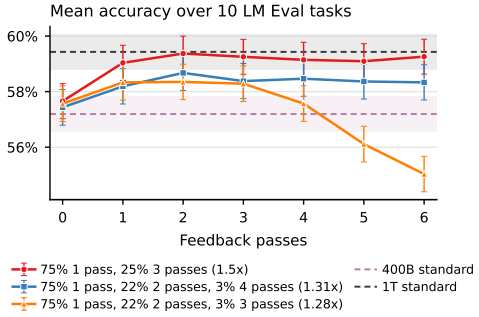
(a) 200B tokens



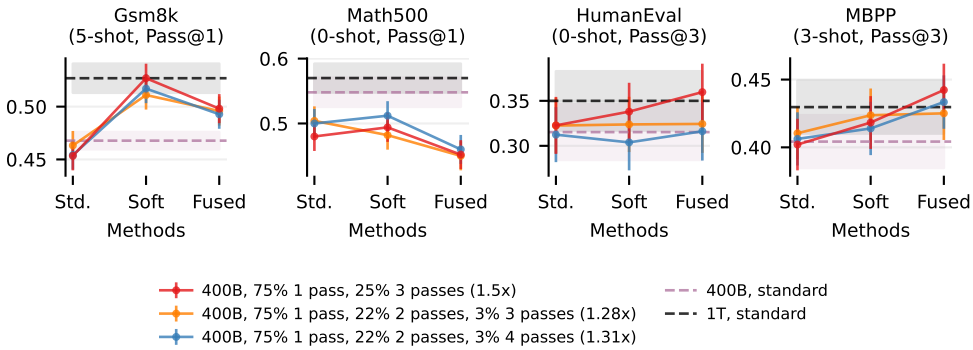
(b) 400B tokens



(c) Similar to Fig. 3 and 9a, but extending number of feedback pass to 1,000. The extrapolation remains stable far beyond the 3 passes used in training time.



(d) LM Eval results, 400B tokens scale



(e) Free form generation results, 400B tokens scale

I Comparison of LM eval performance with other models of similar scale

Model Name	Tokens	W/G	PIQA	OBQA	ARC-E	ARC-C	Avg.
OPT 1.3B	300B	59.59	72.36	33.40	50.80	29.44	49.87
Pythia 1B	300B	53.43	69.21	31.40	48.99	27.05	46.21
Pythia 1.4B	300B	57.38	70.95	33.20	54.00	28.50	49.34
TinyLlama 1B	2T	59.43	73.56	36.80	55.47	32.68	53.23
Llama3.2 1B	9T	60.46	74.54	37.00	60.48	35.75	55.31
Qwen3 1.7B	36T	61.01	72.36	36.80	69.91	43.26	57.30
EvoLM 1B (Qi et al., 2025)	20B	51.30	67.85	32.80	54.80	29.61	46.44
	40B	54.62	69.59	36.20	58.08	30.29	49.38
	80B	53.59	70.78	37.20	62.71	35.92	51.88
	160B	53.99	71.71	36.60	63.09	36.09	52.30
	320B	53.51	71.93	37.20	62.29	36.18	52.49
Full-bandwidth transformer 1B	200B (0 feedback pass)	60.46	71.11	34.60	62.42	34.73	52.66
	200B (1 feedback pass)	62.59	71.49	35.00	63.43	35.41	**53.58**

Table 2: 0-shot LM Eval performance comparison, numbers for EvoLM and other open-sourced models are adopted from Table 4 in the appendix of Qi et al. (2025).

J Additional evaluation results for instruction-tuned model

We additionally evaluate the model on instruction following benchmark, IFEval (Zhou et al., 2023) and the English subset of WritingBench (Wu et al., 2026), using the official fine-tuned Qwen 2.5 7B as the critic model.

For IFEval, we use greedy decoding, and for WritingBench, we use the official suggested random decoding setting, with temperature 0.7, top-p of 0.8 and top-k of 20.

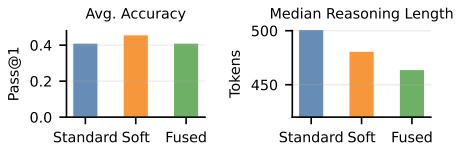
The results are shown in table 3. Broadly, we did not see full-bandwidth models differ from standard model in these two evaluation settings.

	Standard Transformer		Full-bandwidth, 400B		
Task	400B	1T	Std.	Soft	Fused
WritingBench	42.0	42.7	42.04	42.51	42.90
IFEval (prompt strict)	34.94	36.60	35.30	34.57	34.75
IFEval (instruction strict)	45.44	45.92	46.76	45.08	44.60

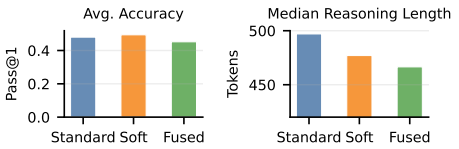
Table 3: Some other results.

K CONCISE REASONING

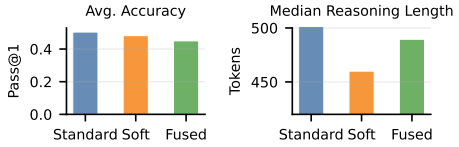
Fig. 6 in the main text demonstrates the shortened rollout length for the 200B tokens model. Same observations on SOFT and FUSED shortening reasoning than STANDARD also hold on other 400B tokens scale run under the three different schedule considered in Sec. H, shown in Fig. 10 below.



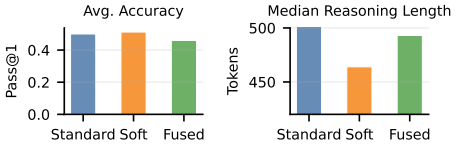
(a) 200B, same as Fig. 6.



(b) 400B, 1.5×.



(c) 400B, 1.28×.



(d) 400B, 1.3×.

Figure 10: Reasoning length and accuracy on 0-shot MATH-500 across different runs.

L GENERATION LENGTH

In the table below, we summarize the median generation length, aggregated over all samples in the benchmark dataset. Note that the length is far beyond the three-token horizon the full bandwidth model is exposed to during training, confirming that at inference time, the model manages to extrapolate a recurrent horizon way longer than training time.

task	budget	STANDARD (mean/med)	SOFT (mean/med)	FUSED (mean/med)
gsm8k	512	242.1 / 231	234.5 / 224	239.9 / 233
math500	2048	619.5 / 486	607.2 / 467	600.4 / 471
humaneval	1024	131.7 / 96	137.8 / 108	143.8 / 119
mbpp	1024	84.3 / 42	78.4 / 43	72.2 / 44
ifeval	1536	400.0 / 279	375.4 / 256	376.3 / 256
WritingBench	4096	688.8 / 690	643.2 / 679	643.2 / 679

Table 4: Generation length statistics for each task (mean / median tokens). Top 4 tasks are based on base model whereas the last two tasks are based on instruction tuned model.

M NUMBER OF PASS V.S. FREE FORM RESULTS

For the 400B token $1.5\times$ run in table. 1, we evaluated FUSED under additional prefill pass. The results are shown below, where we do not observe further improvement. Notably, the performance does not peak at the number of passes used at training time.

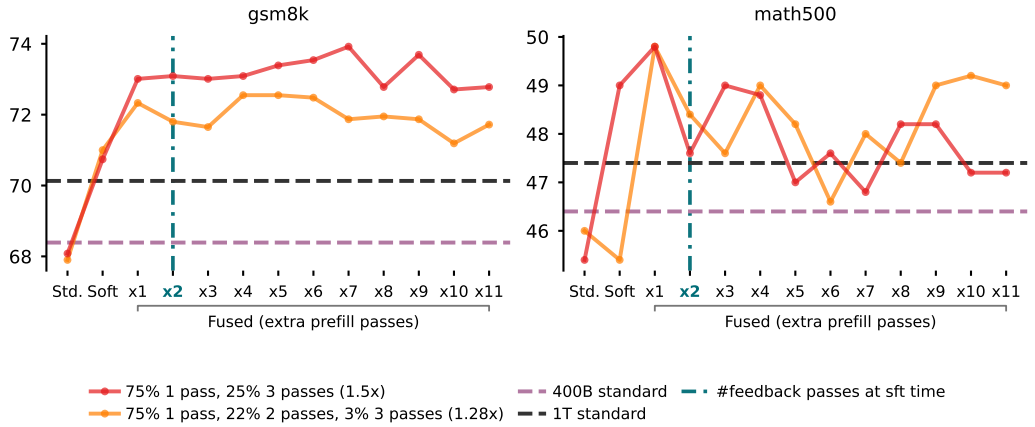


Figure 11: **Increasing the number of prefill pass does not further improve FUSED's performance.**

N POST-TRAINING AND LIKELIHOOD ESTIMATION FOR OFF-POLICY ROLLOUTS

A common post-training procedure samples rollouts from an off-policy sampler and evaluates them under the policy being optimized. The policy gradient then depends on the rollout likelihood, typically weighted by an advantage and an importance ratio.

For on-policy rollouts, likelihood evaluation is essentially free: token log-probabilities can be recorded during decoding. For an off-policy rollout, however, the latent states under the policy model are not available and must be reconstructed. Under latent-feedback decoding, the input at position t depends on the previous policy hidden state,

$$\boldsymbol{u}_t = \boldsymbol{e}_t \otimes \boldsymbol{h}_{t-1}. \tag{12}$$

Crucially, $\boldsymbol{h}_{t-1}$ cannot be obtained directly from the token prefix alone: it is produced by running position $t-1$ with $\boldsymbol{h}_{t-2}$ as latent feedback,

$$\boldsymbol{h}_{t-1} = f_\theta(\boldsymbol{e}_{t-1} \otimes \boldsymbol{h}_{t-2}; C_{t-1}). \tag{13}$$

Thus, to evaluate token t, the policy must first replay positions $1, \ldots, t-1$ to reconstruct its own latent trajectory. Consequently, denote the rollout as $y_{1:T}$ and the prompt as x, we have

$$\log \pi_\theta(y_{1:T} \mid x) = \sum_{t=1}^{T} \log \pi_\theta(y_t \mid x, y_{<t}, h_{t-1}), \tag{14}$$

which cannot be evaluated by standard parallel teacher forcing, but instead requires $O(T)$ sequential steps.

Potential solution 1: Hidden-state replay. To avoid sequentially reconstructing the latent trajectory, one can directly reuse the hidden states produced by the behavior policy during sampling, analogous to router replay in MoE post-training (Ma et al., 2025). Specifically, instead of recomputing the policy hidden state $\boldsymbol{h}^{\pi}_{t-1}$, we approximate

$$\boldsymbol{h}^{\pi}_{t-1} \approx \boldsymbol{h}^{\mu}_{t-1},$$

where μ denotes the sampler policy, and evaluate

$$\log \pi_\theta(y_t \mid y_{<t}, \boldsymbol{h}^{\mu}_{t-1}).$$

This removes the sequential hidden-state replay cost, but introduces bias whenever the latent trajectories of the behavior and policy models differ.

Potential solution 2: Multi-forward pass Let the exact latent-feedback trajectory satisfy

$$\boldsymbol{h}_t^\star = F_{\theta,t}(\boldsymbol{e}_t, \boldsymbol{h}_{t-1}^\star; C_t), \tag{15}$$

which requires sequential replay. Instead, starting from parallel states $\{\boldsymbol{h}_t^{(0)}\}$, we perform K Jacobi-style refinement passes,

$$\boldsymbol{h}_t^{(k)} = F_{\theta,t}(\boldsymbol{e}_t, \boldsymbol{h}_{t-1}^{(k-1)}; C^{(k)}), \qquad k = 1, \ldots, K,$$

where all positions are updated in parallel within each pass. Unrolling gives

$$h_t^{(K)} = F_{\theta,t}\Big(\boldsymbol{e}_t, F_{\theta,t-1}\big(\boldsymbol{e}_{t-1}, \ldots, F_{\theta,t-K+1}(\boldsymbol{e}_{t-K+1}, \boldsymbol{h}_{t-K}^{(0)}; C_{t-K+1}^{(0)})\big); C_t^{(K)}\Big), \tag{16}$$

so K passes explicitly recover K hops of the recurrent dependency, while longer-range information is represented only through the initialization $\boldsymbol{h}_{t-K}^{(0)}$. We then approximate

$$\log \pi_\theta(y_{1:T} \mid x) \approx \sum_{t=1}^{T} \log \pi_\theta\Big(y_t \mid x, y_{<t}, h_{t-1}^{(K)}\Big).$$

Thus, K controls a finite-horizon approximation to the fully autoregressive latent-feedback distribution, reducing from $O(T)$ sequential steps to $O(K)$ parallel passes over the sequence.

An empirical estimation of the approximation error is shown below in Fig. 12, where one forward pass denotes just using the embedding as input. The approximation error reduces significantly after two forward passes and plateaus after three passes.

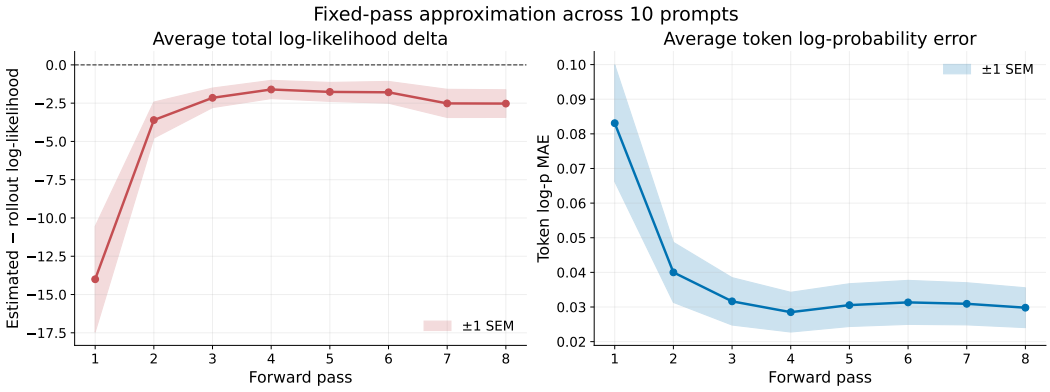


Figure 12: True sequence likelihood can be approximated with multiple forward passes rather than going over all tokens.

O INTERACTION OF FULL-BANDWIDTH TRANSFORMER WITH SLIDING WINDOW ATTENTION

Sliding-window attention (SWA) reduces the KV-cache and attention cost by restricting each token to a local context, but correspondingly limits direct access to distant hidden states. As illustrated in Fig. 13 left, under standard decoding a token can only access hidden states that fall within the SWA receptive field. Information from more distant tokens must therefore propagate through successive local-attention layers.

Full-bandwidth decoding introduces an additional recurrent communication path, which enables information that has reached $\boldsymbol{h}_{t-1}^{L}$ to continue to propagate even after its originating token falls outside the attention window:

$$\boldsymbol{h}_{t-k}^{L} \to \boldsymbol{h}_{t-k+1}^{L} \to \cdots \to \boldsymbol{h}_{t-1}^{L} \to \boldsymbol{u}_t. \tag{17}$$

Thus, SWA provides high-bandwidth local memory through the KV cache, while latent feedback provides a persistent, compressed channel for information beyond the local window. Importantly, latent feedback does not recover individual hidden states that have left the window; instead, their information can be summarized and carried forward through the recurrent hidden state.

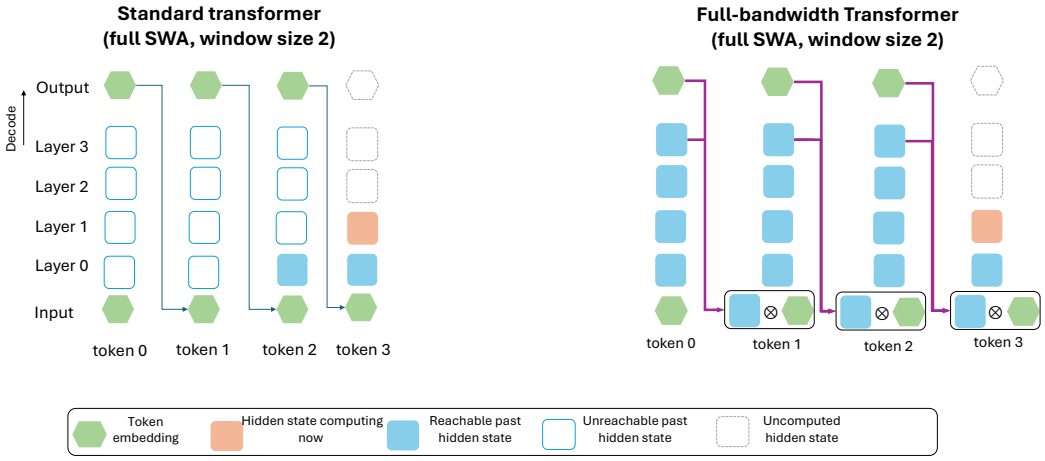


Figure 13: **Full-bandwidth decoding complements sliding-window attention.** Consider an idealized SWA-only model; standard decoding can access only hidden states within the local receptive field (left). Full-bandwidth decoding additionally propagates the previous top-layer state through latent feedback (right), providing a recurrent path for information to persist beyond the attention window.

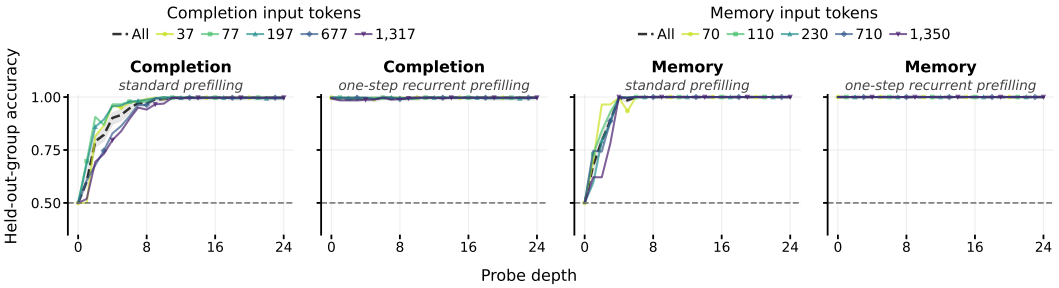
P INFERENCE LATENCY

In the actual vLLM implementation, the extra overhead comes from two parts: The host side for moving around hidden states and the runner side from the extra glu cross gate.

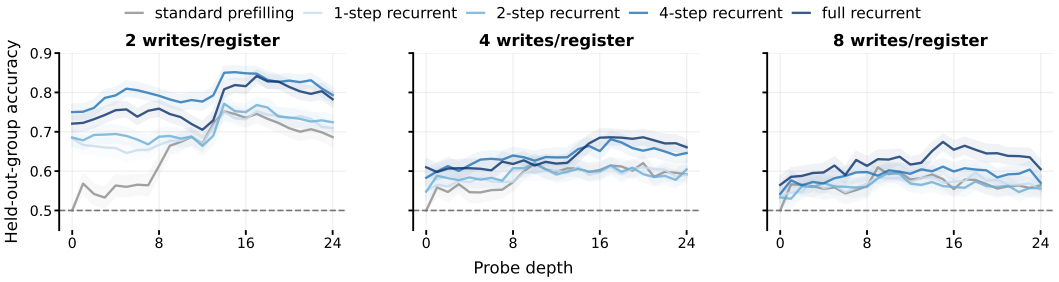
In the end-to-end benchmark under batch size of 1, performed on one H100, latent feedback decoding reduces the throughput by only 2 percent.

STD. tok/s	SOFT tok/s	SOFT/STD.	slowdown	STD. ms/step	SOFT ms/step
214.1	209.1	0.98	2.3%	4.670	4.782

Table 5: Throughput and per-step latency comparison.



(a) **State retrieval.** Each sequence specifies either a binary relation between two counters ("Completion") or a stored absolute binary value ("Memory"), followed by varying number (denoted by line color) of label-independent distraction tokens. One recurrent step makes the target state nearly perfectly decodable at layer 0 across input lengths, whereas standard prefilling requires multiple layers to reconstruct it from the prefix.



(b) **Multi-register latest-write tracking.** Each sequence performs 2,4, or 8 writes to each of eight binary registers and then queries one register's latest value. We probe its value at each residual depth. Recurrent prefilling improves shallow standard accessibility over standard prefilling; One recurrent step's gain diminishes in deeper layers and in inputs with more overwriting, where full recurrent performs the best, showing the benefit of maintaining state throughout the sequence.

Figure 14: **Full-bandwidth transformer exposes global state to shallow layers.** On three synthetic tasks, we linearly probe the final input token's residual stream across depth (0 denotes input) to predict a binary state of the input (See Appendix R for details); We compared standard prefilling, which uses token embedding as inputs, with recurrent prefilling, where the preceding token's top-layer state is fused into the current token's input, similar to Eq. (6) but uses input token rather than sampled token; k-step recurrence applies this fusion over the final k tokens (at the cost of $k+1$ forward pass), while full recurrence applies it throughout the task sequence (at the cost of fully sequential prefill).

Q Full-bandwidth transformer carries richer information in shallow-layer residuals

To verify the added bandwidth directly, we run controlled state-tracking experiments in which the target is fixed but the intervening context varies (full construction in App. R). Two tasks isolate the effect. *Completion tracking* asks whether a completed counter has reached a required one after a run of no-op updates; *delayed memory* asks the model to recover an initial binary state after a sequence of label-independent scratch operations. Both end at a shared colon, and the label is determined entirely by information before it, so a probe at that colon measures how much of the global state each layer has already reconstructed.

We compare two prefilling regimes. Under *standard prefilling*, the final token enters as its plain embedding; under *one-step recurrent prefilling*, that embedding is fused with the preceding token's top-layer state (Eq. (4)), exactly the layer-0 input latent feedback supplies at decode time. We then fit a linear probe for the target (DONE/MORE or ZERO/ONE) at each residual-stream depth.

The two regimes differ sharply at the bottom of the stack. Under standard prefilling, a shallow residual can read only the layer-matched, partially processed prefix (the reachability constraint of Eq. (2)), so reconstructing the global state takes several layers of further computation; the layer-0 probe is near chance. Recurrent prefilling instead exposes a fully processed prefix summary at the layer-0 input, and layer-0 probe accuracy rises to 99.6% for completion tracking and 100% for delayed memory.

Recurrence thus provides a high-bandwidth shortcut that transports globally aggregated information into shallow computation, the mechanism the full-bandwidth view predicts.

One caveat bears emphasis: improved *decodability* does not by itself imply improved *output*. That a target is linearly recoverable at layer 0 shows the information is present, not that the model uses it to decide the next token; making state available and causally exploiting it are distinct, and only the downstream task results (Sec. 4) speak to the latter.

R EXPLANATION ON STATE TRACKING TASKS

We construct paired synthetic examples whose label is determined by information appearing before a shared final colon. The target token itself is never included in the input. We append $0, 8, 32, 128,$ or 256 semantically null scratch updates, allowing us to vary sequence length without changing the target. At the final colon, we record the layer-0 input and the output of every Transformer block.

Completion tracking. Each input specifies a required count a and a completed count b. The target is DONE if $a = b$ and MORE otherwise. For each unordered numeral pair $\{a, b\}$, we include all four assignments $(a, a), (a, b), (b, a), (b, b)$, balancing every numeral across fields and labels. A representative matched pair, abbreviated to show eight repeated distractors, is

```
required = 4                    required = 4
completed = 9                   completed = 4
scratch = 7                     scratch = 7
scratch += 0                    scratch += 0
  ... (8 updates)                 ... (8 updates)
Status:                         Status:
```

The left target is MORE, whereas the right target is DONE. The two examples share the required count, scratch context, distractor sequence, and final token; only the relation between the two counters changes.

Delayed memory. Each input first assigns a binary state and then presents label-independent scratch operations. The target is ZERO or ONE according to the initial state. For example,

```
state = 0                       state = 1
scratch = 0                     scratch = 0
scratch ^= 0                    scratch ^= 0
scratch ^= 1                    scratch ^= 1
scratch ^= 1                    scratch ^= 1
scratch ^= 0                    scratch ^= 0
scratch ^= 1                    scratch ^= 1
scratch += 0                    scratch += 0
  ... (8 updates)                 ... (8 updates)
# final state:                  # final state:
```

The corresponding targets are ZERO and ONE. Thus the model must retain the initial bit while processing an identical intervening context. Completion tracking tests a relational state computed from multiple fields, whereas delayed memory tests persistent transport of an already specified state.

Multi-register latest-write tracking. We additionally test whether recurrent prefilling can expose several independently updated variables. An input assigns binary values to registers $r_0, \ldots, r_{m-1}$, performs eight label-independent scratch updates, and then queries one register. The target is ZERO or ONE according to that register's most recent assignment. For example, the following matched inputs share the complete update history and differ only in the queried register:

```
r4 = 0                          r4 = 0
r4 = 1                          r4 = 1
r0 = 1                          r0 = 1
```

```
r7 = 0                          r7 = 0
  ... (10 assignments)            ... (10 assignments)
r7 = 1                          r7 = 1
r1 = 0                          r1 = 0
scratch = 7                     scratch = 7
scratch += 0                    scratch += 0
  ... (7 updates)                 ... (7 updates)
query = r0                      query = r1
Value:                          Value:
```

Here the latest values are $r_0 = 1$ and $r_1 = 0$, so the left target is ONE and the right target is ZERO. The model must therefore preserve the latest value of every register and bind the final query to the appropriate component of that state.

Probe construction. We train an L_2-regularized linear classifier at each residual-stream depth using four-fold grouped cross-validation. Completion splits hold out entire unordered numeral-pair groups, and memory splits hold out complete scratch-context groups. The enlarged experiment contains $1{,}600$ completion examples from 80 groups and $1{,}280$ memory examples from 128 groups. Because every example ends at the same colon token, the standard layer-0 representation contains no label information beyond the shared token embedding; any above-chance accessibility must be introduced by processing the prefix or by recurrent fusion.

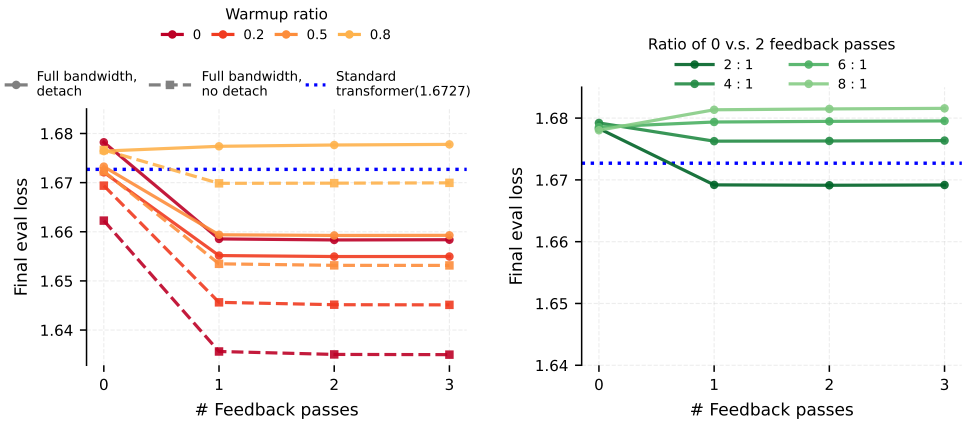
Register-count and overwrite sweeps. In the register-count sweep, every input contains 16 assignments and eight null updates and is padded to exactly 180 tokens; only the number of registers varies over $m \in \{1, 2, 4, 8\}$. This separates the effect of maintaining more variables from input length and total update count. We use 128 structural groups per register count. Each group contains a random register-update schedule, its bitwise value complement, and queries for every register, and grouped cross-validation holds out the entire schedule and all associated queries. The resulting sweep contains $3{,}840$ examples per prefill condition. To vary overwrite interference directly, we then fix $m = 8$ and use $2, 4$, or 8 writes per register. Each setting contains $2{,}048$ examples from 128 groups and produces inputs of 180, 276, and 468 tokens, respectively.

Recurrent-suffix controls. Besides standard and full recurrent prefilling, we recurrently prefill only the final $k \in \{1, 2, 4\}$ input tokens while standard-prefilling the preceding prefix. One step fuses state only at the shared final colon, two steps recurrently process `Value:`, and four steps additionally include the queried-register digit and newline. We probe the residual stream at the final colon at layers $0, 1, 2$, and 4, as well as at every remaining depth, using the same grouped L_2-regularized classifiers. This sweep distinguishes information accumulated throughout the update sequence from information made accessible locally while processing the final query.

S ABLATION STUDY

In this section, we perform ablation studies for several of our design choices.

Detachment of hidden states We consider detaching the hidden states after each feedback pass during training, which reduces memory footprint. The solid lines in Fig. 15a show the results: Models trained under detachment still show improved validation loss in second pass, however their level of improvement is much smaller compared with the non-detached version (dashed lines.)



(a) Detaching the hidden state and longer warmup both reduce the gain of feedback pass. Feedback pass 0 denotes standard prefilling with embedding as inputs.

(b) Interleaving between one and multi-pass does not give more efficient training. The hope is that such interleaved training can reduce the number of multi-pass steps while preserving the performance.

Figure 15: **Ablation study on feedback scheduling and hidden state detachment.** Training a 1B model on 10B tokens, more iterations with multiple forward passes (darker line, shorter warmup ratio) gives better validation loss, for both first pass and the second pass. Having standard training and multi pass training in bulk provides better performance than interleaving two regimes.

Alternative scheduling regime In the current experiments, we introduced feedback passes later on in the training. Lines of different colors in Fig. 15a consider introducing multi forward pass at different stage, where warmup ratio of 0 denotes multi pass from the beginning. Overall, the longer the multi pass stage is, the more improvment we get. We also considered alternative scheduling, where we interleaved between one pass and multi pass at different iterations. Fig. 15b shows that that it fails to give performance on par with having single and multi pass as two separated stages. And a bigger issue is that such interleaved training prevents us from starting from an intermediate checkpoint.

Role of projection matrices The fusion gate (Eq. (4)) contains two projections, $\boldsymbol{W}^H$ and $\boldsymbol{W}^G$, which respectively transform the previous hidden state and the current token embedding. In Fig. 16, we examine whether these projections bottleneck / truncate information or have collapsed outputs. The hidden-state projection $\boldsymbol{W}^H$ remains nearly full rank (1530/1536) with a smoothly decaying singular spectrum, and applying it largely preserves the variance spectrum of the observed hidden states, although low-variance directions are somewhat attenuated. Thus, $\boldsymbol{W}^H$ does not appear to collapse the hidden state into a low-dimensional subspace. Meanwhile, the sigmoid activations $\sigma(\boldsymbol{W}^G \boldsymbol{e}_t)$ span a broad range between zero and one and vary substantially across tokens, suggesting that $\boldsymbol{W}^G$ primarily provides token-dependent modulation rather than a fixed dimensional mask. Together, these results suggest that the fusion gate preserves a high-dimensional channel from the previous hidden state while allowing the current token to selectively reweight it.

Training dynamics We present examples of what the loss curve looks like during training in Fig. 17. The second pass loss (dashed lines) begins above the first pass loss (solid line) and then

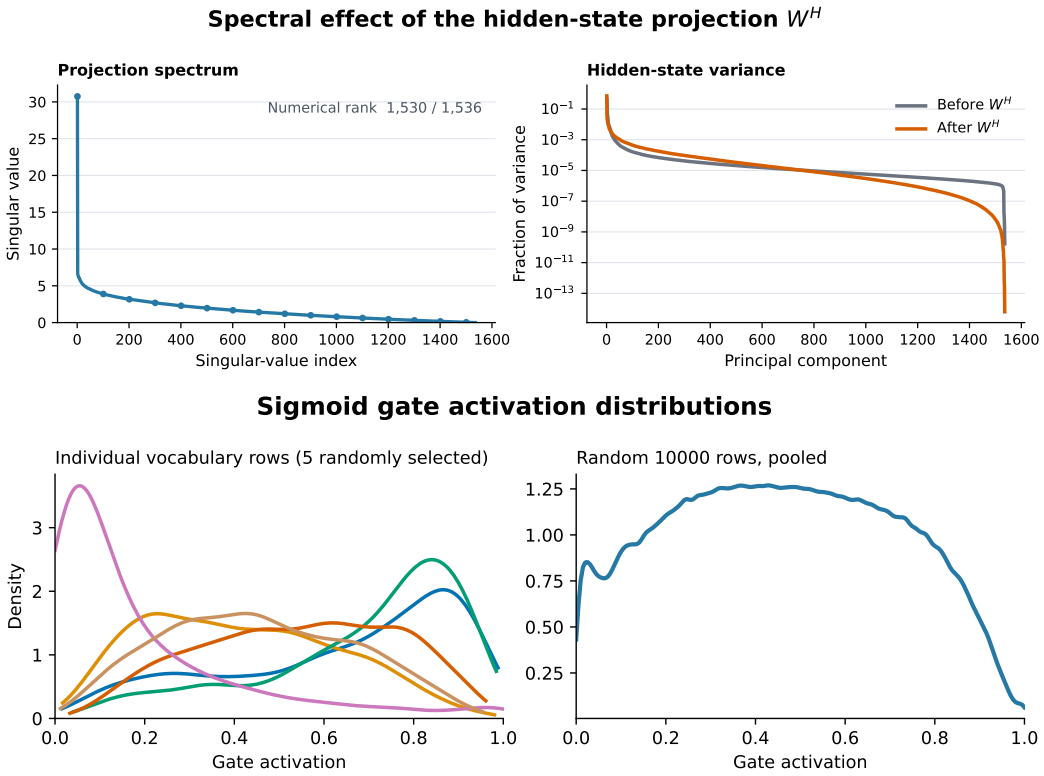


Figure 16: **Inspection of the fusion gate for a model trained on 200B tokens.** *Top:* the hidden-state projection $\boldsymbol{W}^H$ is nearly full rank and largely preserves the variance spectrum of hidden states. *Bottom:* sigmoid gate activations $\sigma(\boldsymbol{W}^G \boldsymbol{e}_t)$ vary broadly across dimensions and tokens, indicating token-dependent modulation of the recurrent state.

gradually goes below the first pass. This pattern is useful for debugging the training: A successful training configuration should result in the dashed lines going under the solid line, which implies that the model learns to read and use its own hidden states for improving the context understanding, a necessary (but insufficient) condition for latent feedback decoding to work.

Role of jitter noise Lastly, we ablate the role of the jitter noise (Eq. (11)). Fig. 18 shows the loss v.s. feedback passes on two models trained on 1B tokens, with and without jitter noise, where we observe that jitter noise slightly improves the loss, especially under standard prefilling. The jitter noise is not observed to change the stable attractor pattern.

S.1 Symmetric gate

Evaluated on a separate smaller-scale setting: A 20-layer Nanochat model (950M parameters) trained on 20B tokens using a modified Nanochat codebase. We consider three types of fusion gates

- Gated product: Same as the glu cross fusion (Eq. (4)) used in the main text.
- Linear addition: $\boldsymbol{W}^H \boldsymbol{h}_{t-1} + \boldsymbol{W}^G \boldsymbol{e}_t$.
- Concatenation and projection:$\boldsymbol{W}^R[\boldsymbol{h}_{t-1}; \boldsymbol{e}_t]$, $\boldsymbol{W}^R \in \mathbb{R}^{D \times 2D}$.

The results are shown in Fig. 19, where we do not see one method outperform another one on a small scale, nor do we observe that symmetric fusion shows signs of shortcut seeking. This may suggest that future large-scale experiments can consider alternative forms of fusion.

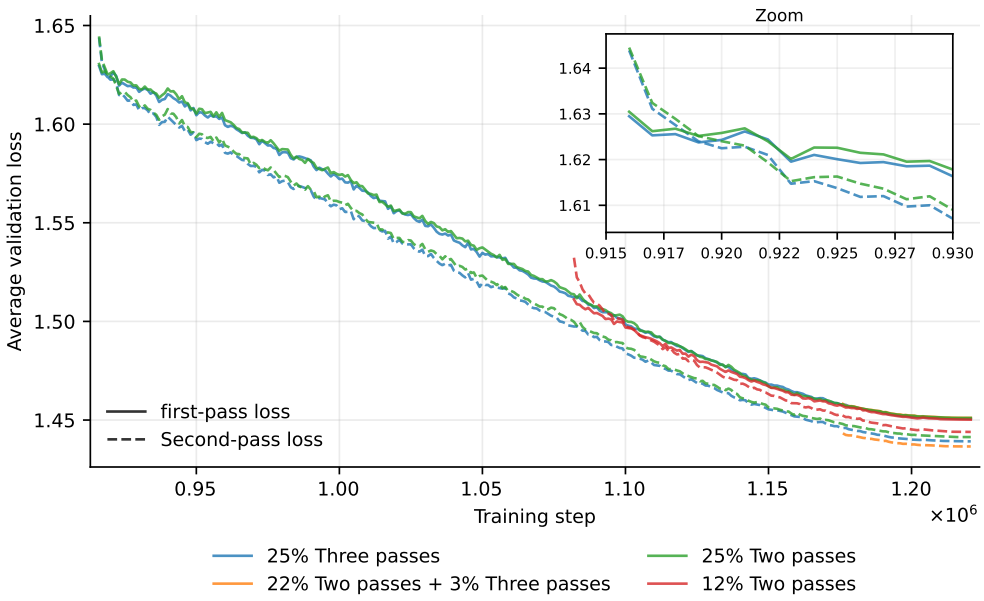


Figure 17: Example training dynamics of full-bandwidth transformer for the 400B token run. The feedback pass is only introduced at the cooldown phase (starting from 75% of the training)

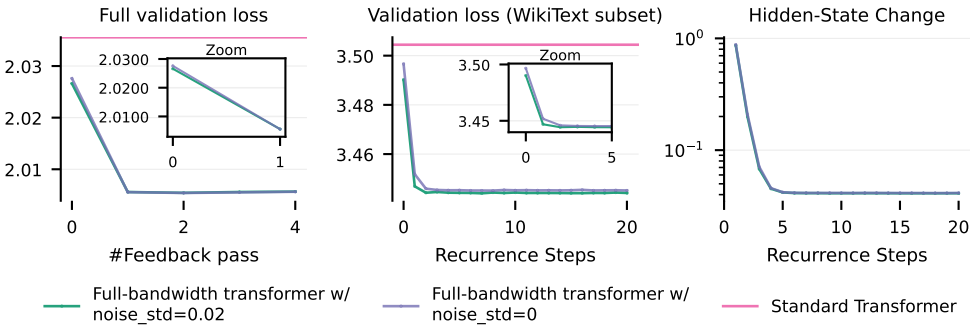


Figure 18: Ablation study on the effect of noise (Eq. (11)). Evaluated on 1B scale run, jitter noise improves the loss, especially on the first pass loss.

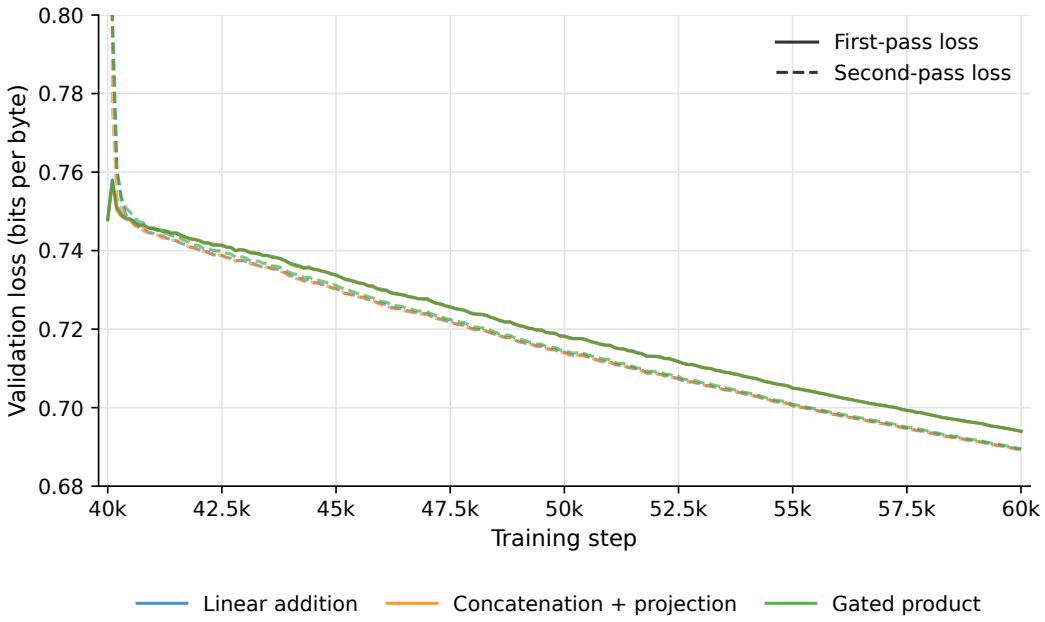


Figure 19: Tested on a small-scale Nanochat run, fused inputs under symmetric gate still manage to improve the validation loss.

Problem

What is the last nonzero digit to the right of the decimal point in the decimal expansion of $\frac{137}{500}$?

Standard decoding (200B model)

To determine the last nonzero digit to the right of the decimal point in the decimal expansion of $\frac{137}{500}$, we start by performing the division of 137 by 500.
First, we set up the division:

$$137 \div 500$$

Since 137 is less than 500, the decimal expansion will start with a zero. *[...326 tokens omitted...]*
Since the remainder is now 0, the decimal expansion of $\frac{137}{500}$ is:

$$0.274$$

The last nonzero digit to the right of the decimal point is $\boxed{4}$.

Soft decoding (200B model)

To find the last nonzero digit to the right of the decimal point in the decimal expansion of $\frac{137}{500}$, we can perform the division and observe the pattern of the decimal expansion.
$\frac{137}{500} = 0.274$
The last nonzero digit to the right of the decimal point in the decimal expansion of $\frac{137}{500}$ is 4.

Figure 20: Qualitative comparison of outputs from the 200B model under standard and soft decoding (greedy decoding). Both reach the correct answer; soft decoding is substantially more concise. Truncated text is indicated by [...]. This is no longer observable on instruction-tuned version.

T MODEL OUTPUT

Soft decoding

**** 200b model, standard decoding ****
To determine the last nonzero digit to the right of the decimal point in the decimal expansion of $\frac{137}{500}$, we start by performing the division of 137 by 500.
First, we set up the division:

$$137 \div 500$$

Soft decoding

**** 200b model, standard decoding ****
To simplify the expression $\frac{(10r^3)(4r^6)}{8r^4}$, we will follow these steps:
1. **Multiply the numerator**:

$$(10r^3)(4r^6)$$

Using the properties of exponents, we can combine the terms:

$$10 \cdot 4 \cdot r^{3+6} = 40r^9$$

2. **Rewrite the expression with the simplified numerator**:

$$\frac{40r^9}{8r^4}$$

3. **Simplify the fraction**: - First, simplify the coefficients:

$$\frac{40}{8} = 5$$

- Next, simplify the exponents of r:

$$\frac{r^9}{r^4} = r^{9-4} = r^5$$

4. **Combine the simplified parts**:

$$5r^5$$

Thus, the simplified form of the expression $\frac{(10r^3)(4r^6)}{8r^4}$ is:

$$\boxed{5r^5}$$

**** 200b model, soft ****
To simplify the expression, we can first simplify the numerator and then divide by the denominator.
Numerator: $(10r^3)(4r^6) = 40r^9$
Denominator: $8r^4$
Now, we divide the numerator by the denominator:
$\frac{40r^9}{8r^4} = 5r^{9-4} = 5r^5$
So, the simplified expression is $5r^5$.