WRITE BACK THE Δ: REVISITING THE SAME TOKENS WITH FRESH REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Transformers process information strictly forward through depth, preventing deeper computation from revisiting and refining earlier representations. To augment the standard forward pass, existing approaches either re-execute depth, incurring additional computation, or modify the residual stream using predefined directions, limiting their instance-level adaptation. Recently, inference-time feedback offers a direct mechanism for recycling endogenously produced computation by writing deeper residual states back to earlier layers, yet what should be fed back remains unclear. We argue that the depth increment Δ, capturing newly accumulated computation between two layers, provides a more effective, composable, and scalable feedback signal than the full state. Building on this observation, we introduce **ReFlux**, a learnable feedback graph that dynamically selects and composes increment-carrying routes. ReFlux supports synchronous feedback to the same token and streaming feedback to subsequent tokens. Extensive experiments across various models, corpora, and benchmarks show that synchronous ReFlux consistently reduces perplexity across ten language-modeling corpora, and improves accuracy by **2.1–2.3 points**, with gains reaching **4.7 points** on multi-hop reasoning. Streaming ReFlux further retains most of these gains while preserving the base model's **1$\times$ theoretical backbone FLOPs**. These results establish ReFlux as an efficient paradigm for unlocking the latent computational potential of LLMs, allowing them to revisit the same tokens with fresh representations.

1 INTRODUCTION

> *"The real voyage of discovery consists not in seeking new landscapes, but in having new eyes."*
> — Marcel Proust, *In Search of Lost Time*

Some of our most valuable insights emerge only after deeper processing: a later distinction or connection can change how earlier evidence should be interpreted. Intelligence, therefore, depends not only on producing new information, but also on reusing it to reshape ongoing interpretation, thereby ***revising how we see with what we have discovered***. Standard decoder-only Transformers lack a native path for this kind of reuse. Their residual stream accumulates layer-wise updates in one direction through depth (Vaswani et al., 2017; He et al., 2016; Elhage et al., 2021). Later layers can build on earlier computation, but what they add cannot return to condition earlier processing.

Existing approaches explore two ways to augment the standard forward computation. Recurrent and looped Transformers *re-execute depth*, obtaining refinement through specialized training or additional inference computation (Dehghani et al., 2018; Zhu et al., 2025; Bae et al., 2026). Activation steering instead *edits the residual stream*, but typically relies on directions extracted in advance from external data or predefined behaviors (Turner et al., 2023; Li et al., 2023; Rimsky et al., 2024; O'Brien et al., 2024), limiting instance-level adaptation (Braun et al., 2025; Jiang et al., 2026). Rather than recomputing depth or applying a preset correction, we advocate for a complementary axis of reuse: ***recycling the computation the model has already produced endogenously.***

Recirculation moves closer to this goal by writing a deeper residual state back to an earlier layer in a frozen language model (Mozer et al., 2026). However, because the residual stream is cumulative, the deep state contains both information already present at the earlier layer and newly accumulated computation, making feedback increasingly redundant as routes accumulate. This raises a sharper question: ***should feedback replay where the model arrived, or only what depth added?***

We answer by making newly accumulated computation the unit of feedback. For a deeper state z_s and an earlier state z_d, we define the **depth increment** as $\mathbf{\Delta}_{d\to s} = z_s - z_d$, which captures the residual computation accumulated between the two layers. Controlled studies show that full-state feedback

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.

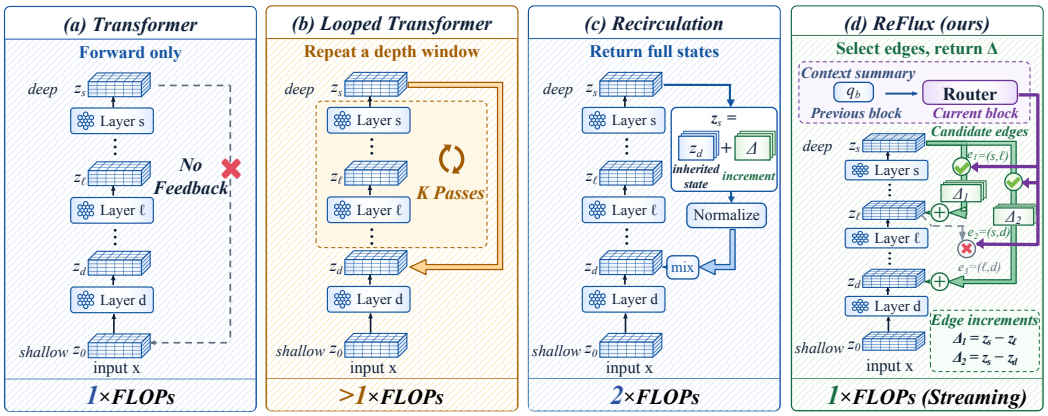


Figure 1: Revisiting earlier representations with new computation. Looped Transformers re-execute depth, while Recirculation returns full states to earlier layers. ReFlux instead isolates the newly accumulated computation and selectively routes depth increments Δ through a learned feedback graph, enabling streaming feedback without increasing the backbone's theoretical FLOPs.

saturates as more feedback is introduced, whereas increment feedback continues to benefit from additional routes and rounds. Moreover, increments from different layer pairs provide complementary gains. Feedback thus becomes an allocation problem: ***which*** increments should be reused, ***where*** should they be written, and with ***what*** strength for the current context?

To make this form of reuse a first-class operation, we propose **ReFlux**, a learnable sparse feedback graph over frozen Transformers (Fig. 1). Each edge carries an increment from a deeper source to a shallower target, while a lightweight router selects the active edges and their strengths conditioned on the current context. The same graph supports two execution schedules: **synchronous** execution revisits a token using its own deeper computation, whereas **streaming** execution carries the interpretation accumulated over the processed prefix forward to shape subsequent tokens within a single traversal.

Across five models from two families, ten language-modeling corpora, and eight reasoning benchmarks, synchronous ReFlux consistently improves language modeling performance, yielding systematic perplexity reductions over the base models and prior feedback baselines. It also improves average reasoning accuracy by **2.1–2.3 points**. Gains are largest on multi-hop tasks, averaging **4.7 points** where answers require evidence to be composed across distributed information. Streaming ReFlux retains most of these improvements at the base model's **1× theoretical backbone FLOPs** and only **1.02–1.08× decoding runtime**. Routing analyses further show that ReFlux remains sparse while adapting its selected support to input difficulty and task family.

Together, ReFlux turns depth from a one-way computation path into an input-adaptive source of reusable updates, allowing the model to take a second look at the same tokens with information revealed later. Our contributions are summarized as follows:

- ❶ ***Discover.*** We identify the depth increment Δ as an effective and scalable feedback signal, turning newly accumulated computation into a reusable and composable unit that scales across different dimensions.
- ❷ ***Build.*** We propose ReFlux, a learnable feedback graph over frozen Transformers that dynamically composes increment-carrying routes, with a streaming execution scheme that preserves the base model's $1\times$ theoretical backbone FLOPs.
- ❸ ***Validate.*** We demonstrate consistent gains across five models, ten corpora, and eight benchmarks, including improved language modeling and a 4.7 point gain in multi-hop reasoning, while streaming execution preserves most of these benefits at only 1.02–$1.08\times$ decoding runtime.

2 Related Work

2.1 Recurrent and Looped Transformers

Adding effective depth through layer iteration is a long-standing direction in language-model research. Dehghani et al. (2018) introduced the Universal Transformer, which shares a block across recurrent steps, while Geiping et al. (2026) extend this paradigm to billion-parameter models. Looped transformers (Yang et al., 2024a; Giannou et al., 2023) and looped language models (Zhu et al.,

2025; Kohli et al., 2026) use recurrent depth for algorithmic computation and parameter efficiency, while SMELT (Wang et al., 2026a) scales compute-matched looping to 54B-parameter MoE models. The same principle extends to latent reasoning, with Coconut (Hao et al., 2024), PonderLM (Zeng et al., 2026), implicit chain-of-thought (Deng et al., 2023), Quiet-STaR (Zelikman et al., 2024), and TRM (Jolicoeur-Martineau, 2025) exploring recurrent or recursive latent computation. Adaptive computation further learns where or when to allocate computation, as explored by ACT (Graves, 2016), PonderNet (Banino et al., 2021), MoD (Raposo et al., 2024), and MoR (Bae et al., 2026). These approaches either train pretrained models to introduce recurrence or feedback, as in Encode–Think–Decode (Koishekenov et al., 2025) and FBT (Wang et al., 2026b), or operate training-free by re-executing existing layers, as in TF-Loop (Chen et al., 2026) and CoLa (Li et al., 2025). Thus, adding depth to a checkpoint still requires either training or extra inference FLOPs. ReFlux instead treats increments already computed by the network as reusable payloads and turns their selection into a routing problem, enabling effective depth without additional backbone FLOPs in streaming.

2.2 Inference-Time Interventions on Residual Stream

In parallel, another family of methods edits the residual stream directly at inference time. Activation steering adds behavior-specific directions to internal activations, with prior work exploring directions derived from manually constructed signals (Turner et al., 2023), labeled or paired behaviors (Li et al., 2023; Ye et al., 2026; Rimsky et al., 2024), and representation features (O'Brien et al., 2024). However, because these directions are typically extracted in advance, they may adapt poorly to individual inputs (Braun et al., 2025), suffer from limited generalization beyond the extraction distribution (Jiang et al., 2026), or induce undesirable shifts in activation distributions (Stickland et al., 2024). Patchscopes (Ghandeharioun et al., 2024) re-enters the model with its own representation for readout, rather than feeding it back into the computation that produced it. More recently, Recirculation (Mozer et al., 2026) introduces a feedback mechanism that writes a deeper residual state back into a shallower stream. ReFlux changes what and how information is propagated: it feeds back the depth increment produced by the model itself through a learned feedback mechanism.

Extended related works and a comparison of ReFlux with these methods (Table 6) are in Appendix A.

3 The Δ: A Unit of New Computation

3.1 Deep Computation as Feedback

Consider a decoder-only Transformer that repeatedly updates a shared residual stream. Let $z_\ell(t) \in \mathbb{R}^D$ denote the residual stream for token t after layer ℓ. Each block adds a layer-specific update:

$$z_{\ell+1}(t) = z_\ell(t) + F_\ell(z_\ell, t). \tag{1}$$

Over depth, these updates can integrate broader context and form distinctions unavailable near the bottom of the stack. However, computation is directional across depth: a shallow state can be refined by later blocks, but later interpretations cannot directly revise earlier states. Once carried forward, a block's computation only affects downstream computation, with no route back to the earlier stream.

Recirculation introduces precisely such a feedback path in a frozen language model. Given a source layer s and a target layer d with $s > d$, it writes the deep residual state back to the target stream:

$$\widetilde{z}_d(t) = \beta z_d(t) + \alpha \mathcal{N}_{z_d(t)}\big(z_s(t)\big), \qquad \mathcal{N}_h(z) = \frac{\|h\|_2}{\|z\|_2 + \varepsilon} z. \tag{2}$$

Here, α controls the feedback strength, β retains the target state, and $\mathcal{N}_h$ matches the source state to the target scale. In Recirculation, Eq. 2 uses the full source state z_s, which raises a natural question: *is the full state the right signal to write back?*

3.2 Uncovering the Δ

To answer this question, we first examine what is actually new at each depth. Specifically, we characterize how residual states change across depth under the same sampling protocol as Mozer et al. (2026), using 1,000 fixed-length windows of 1,024 tokens from the C4 validation split (Raffel et al., 2020). For each window, we record the block residual stream and compute the pairwise cosine similarity between depths d and s:

$$C_{d,s} = \mathbb{E}_t \left[\cos\big(z_d(t), z_s(t)\big)\right].$$

We average over token positions within each subset and then aggregate across subsets. We also measure the norm-relative displacement $R_{d,s} := \mathbb{E}_t[\|z_s(t) - z_d(t)\|_2 / (\|z_d(t)\|_2 + \varepsilon)]$. For layer

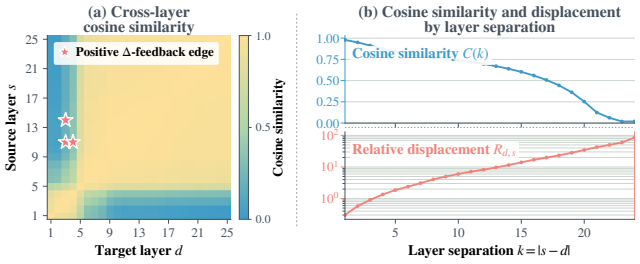


Figure 2: Cross-depth geometry of Gemma3-1B residual states. (a) Pairwise cosine similarity between z_d and z_s; stars mark top-three positive source-target pairs. (b) Cosine similarity and norm-relative displacement versus layer separation k. Adjacent states are similar, while successive layerwise rewrites accumulate into a larger change over longer depth intervals.

Table 1: Matched-payload decomposition under architecture-specific stable feedback protocols on C4, with 64 windows of 512 tokens for each model. We use the best-performing source–target configuration. For Gemma3-1B, we use s=11 → d=4, α=0.25, β=0.9; for Gemma3-4B, s=18 → d=9, α=0.15, β=0.9; for Gemma3-12B, s=35 → d=16, α=0.15, β=1.0; for Qwen3-4B, s=26 → d=13, α=0.2, β=1.0.

	GEMMA3-1B		GEMMA3-4B		GEMMA3-12B		QWEN3-4B	
Payload	PPL ↓	Δ logPPL ($\times 10^{-2}$) ↑	PPL ↓	Δ logPPL ($\times 10^{-2}$) ↑	PPL ↓	Δ logPPL ($\times 10^{-2}$) ↑	PPL ↓	Δ logPPL ($\times 10^{-2}$) ↑
None	$30.772_{\downarrow 0.00}$	0.00	$23.693_{\downarrow 0.00}$	0.00	$42.973_{\downarrow 0.00}$	0.00	$25.562_{\downarrow 0.00}$	0.00
Full z_s	$29.699_{\downarrow 3.49}$	+3.55	$22.790_{\downarrow 3.81}$	+3.89	$41.462_{\downarrow 3.52}$	+3.58	$24.626_{\downarrow 3.66}$	+3.73
Base z_d	$31.110_{\uparrow 1.10}$	-1.09	$23.899_{\uparrow 0.87}$	-0.87	$43.451_{\uparrow 1.11}$	-1.11	$25.599_{\uparrow 0.14}$	-0.14
$\Delta_{d\to s}$	**29.210**$_{\downarrow 5.08}$	**+5.21**	**21.856**$_{\downarrow 7.75}$	**+8.07**	**40.231**$_{\downarrow 6.38}$	**+6.59**	**24.030**$_{\downarrow 5.99}$	**+6.18**

separation k, we use the depth-averaged profiles

$$C(k) := \frac{1}{L-k}\sum_{\ell=1}^{L-k} C_{\ell,\ell+k}, \qquad R(k) := \frac{1}{L-k}\sum_{\ell=1}^{L-k} R_{\ell,\ell+k}.$$

Here L is the number of analyzed block states. Fig. 2(b) shows that adjacent states are highly similar, yet the layerwise rewrites accumulate over depth. Fig. 2(a) gives the corresponding layer-pair view. The three marked best pairs were obtained from an independent sweep over source-target choices. They lie away from the high-similarity diagonal and connect shallow targets to substantially deeper sources, indicating that useful feedback emerges after multiple layerwise rewrites have accumulated.

The deeper state is therefore not just a later copy of the target state. It mixes inherited content with the computation accumulated between the two depths. We therefore distill the depth-wise increment

$$\Delta_{d\to s}(t) := z_s(t) - z_d(t). \tag{3}$$

To isolate its role, we compare three matched payloads: the full source state z_s, the target state z_d, and the increment in Eq. 3. Within each model, the frozen model, source–target pair, injection positions, feedback schedule, feedback strength, and scale protocol are held fixed, only the payload changes. We report perplexity, relative perplexity change, and the mean token-level NLL reduction. As Table 1 shows, the increment is consistently best on architectures with different sizes and normalization schemes. Writing back the target state provides little to no improvement, while the full source state yields smaller gains than the isolated increment, indicating that the feedback benefit lies not in the state itself, but in the change induced by deeper computation.

3.3 SCALING BEYOND SATURATION

We next ask whether this Δ advantage survives when feedback is scaled along three axes: the number of rounds r, the graph width k, and the context length T. Here, r and k denote the number of feedback times and active feedback edges. The round and width sweeps evaluate MMLU accuracy (Hendrycks et al., 2020) and BigPatent perplexity (Sharma et al., 2019), using 2,000 randomly sampled examples; the length sweep measures mean ΔPPL on 50 PG19 validation set books (Rae et al., 2019). We first rank all source–target layer pairs by validation improvement and use nested top-k sets $\mathcal{E}_k = \{e_1, \ldots, e_k\}$ for the width sweep. We evaluate $r \in \{0, 1, 2, 3, 4, 6\}$ feedback rounds and $k \in \{0, 1, 2, 3, 4, 6\}$ active edges. For the length sweep, we use $T \in \{512, 1{,}024, 2{,}048, 4{,}096, 8{,}192\}$. Across the width and length experiments, we keep the total feedback strength $A_{\text{total}} = \sum_{e\in\mathcal{E}} \alpha_e$ fixed at 0.5. For the width sweep, this strength is split equally across edges. We report the length result as $G_{\text{PPL}}(T) = 100\left(1 - \frac{P_{\text{fb}}(T)}{P_0(T)}\right)$, so values near zero indicate that feedback no longer changes perplexity materially.

The round sweep in Fig. 3(a) probes the *depth* of feedback, asking how much iterative refinement can be usefully applied. Full-state feedback gives an early gain but levels off quickly, whereas Δ feedback

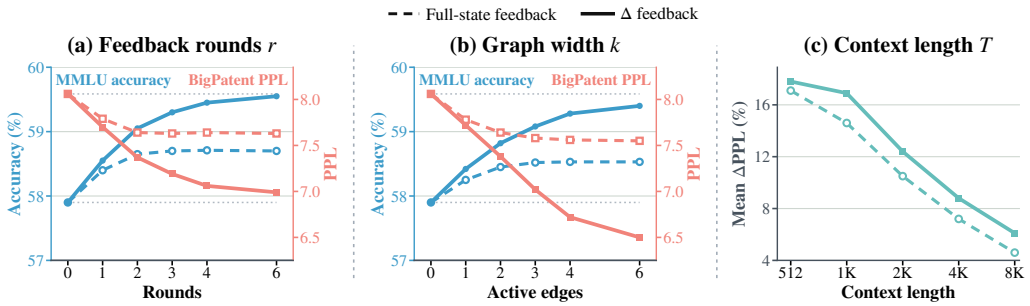


Figure 3: Scaling behavior of feedback across three axes on Gemma3-4B.

continues to improve over a wider range of r. Repeated refinement is therefore more effective when each write carries newly accumulated computation.

The width sweep in Fig. 3(b) probes the *breadth* of feedback, asking how many source–target routes can be combined. Here k is the number of active edges, carrying the source state or depth increment. Full-state feedback saturates after a few edges, whereas Δ continues to benefit from a wider graph, suggesting that increment payloads compose more cleanly across routes.

The context sweep in Fig. 3(c) probes how feedback behaves as *length* grows. As longer prefixes provide the backbone with more usable context, the marginal headroom for a fixed-strength intervention decreases, and the relative gain declines for both protocols. Δ retains more of its short-context benefit, with its performance decaying more slowly as context length increases.

Taken together, Δ feedback retains useful headroom when scaled in depth, breadth, and length. These axes imply different ways to expand feedback: increasing r enables additional rounds of refinement but requires further stack traversals, whereas increasing k introduces more increment-carrying routes without altering the refinement schedule. We therefore focus on the compositional capacity of feedback routes at a fixed refinement depth, where increasing k provides a natural and inexpensive benefit. A static intervention, however, fixes its routes and strengths in advance, leaving it unable to select and combine increments according to the current context. This motivates a more flexible and adaptive formulation in which the available feedback budget can be dynamically allocated across increment-carrying routes.

4 REFLUX

ReFlux instantiates this idea as a feedback graph over a frozen decoder-only Transformer, as illustrated in Fig. 4. Candidate edges connect deeper source states to shallower target states and carry the increments accumulated between the two depths. Let $\mathcal{T}_b$ denote the token positions in routing block b. The router produces one route for each block, shared by all $t \in \mathcal{T}_b$.

4.1 INCREMENT EDGES

Let the frozen Transformer have depth nodes $V = \{0, \dots, L\}$, one node per residual-stream depth. We define a candidate feedback graph $\mathcal{G}_C = (V, \mathcal{C})$, where each directed edge $e = (s_e, d_e) \in \mathcal{C}$ satisfies $s_e > d_e$ and connects a deeper source depth to a shallower target depth. For a token position t, let $h_d(t)$ denote the residual state at target depth d immediately before the write. Given reference states z^{ref} recorded from a traversal, the payload carried by edge e is the depth increment

$$p_e(t) = \Delta_e(t) = z^{\text{ref}}_{s_e}(t) - z^{\text{ref}}_{d_e}(t). \tag{4}$$

The execution schedule determines which traversal supplies these reference states. The edge therefore exposes the computation accumulated over the interval $(d_e, s_e]$.

For block b, the router activates a subset $\mathcal{E}^{(b)} \subseteq \mathcal{C}$. For target depth d, let $\mathcal{I}^{(b)}_d = \{e \in \mathcal{E}^{(b)} : d_e = d\}$ denote its active incoming edges. Given nonnegative edge strengths $\alpha^{(b)}_e$, ReFlux writes the selected payloads additively into the target stream at position t:

$$\bar{h}_d(t) = h_d(t) + \sum_{e \in \mathcal{I}^{(b)}_d} \alpha^{(b)}_e \mathcal{N}_{h_d(t)}(p_e(t)), \qquad \mathcal{N}_h(u) = \frac{\|h\|_2}{\|u\|_2 + \varepsilon} u. \tag{5}$$

Since $\mathcal{N}_{h_d}$ matches the payload norm to that of the target state, the write is determined by both its strength and the direction of the carried payload. The specific normalization protocols are in App. B.

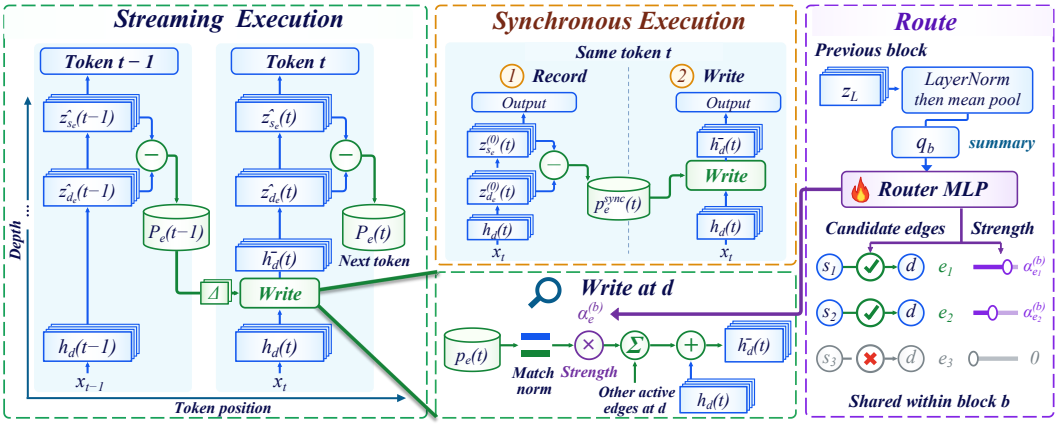


Figure 4: Overview of ReFlux Architecture and Execution Schedules.

4.2 Learning to Route

Once the payload is fixed, routing amounts to selecting a weighted subgraph of $\mathcal{G}_C$. For each block b, every candidate edge receives a binary gate and a nonnegative amplitude,

$$m_e^{(b)} \in \{0,1\}, \qquad a_e^{(b)} \geq 0, \qquad \alpha_e^{(b)} = m_e^{(b)} a_e^{(b)}. \tag{6}$$

The gate specifies the support of the route, and the amplitude specifies the write strength on each selected edge. After thresholding, the active graph and its realized width are

$$\mathcal{E}^{(b)} = \{e \in \mathcal{C} : m_e^{(b)} = 1\}, \qquad \kappa_b = |\mathcal{E}^{(b)}|. \tag{7}$$

For each block, the router uses a layer-normalized, mean-pooled residual state from the preceding block as its causal context summary q_b. A lightweight router g_ϕ produces gate and amplitude logits,

$$(\eta^{(b)}, \rho^{(b)}) = g_\phi(q_b), \qquad a^{(b)} = \mathrm{softplus}(\rho^{(b)}). \tag{8}$$

The discrete support is optimized through a Gumbel-Sigmoid relaxation. During training, Eq. 5 uses the relaxed strengths

$$\tilde{m}^{(b)} = \mathrm{GumbelSigmoid}(\eta^{(b)}, \tau), \qquad \tilde{\alpha}_e^{(b)} = \tilde{m}_e^{(b)} a_e^{(b)}, \tag{9}$$

in place of $\alpha_e^{(b)}$. At deployment, the gate is thresholded,

$$m_e^{(b)} = \mathbb{1}\{\sigma(\eta_e^{(b)}) > \xi\}, \qquad \alpha_e^{(b)} = m_e^{(b)} a_e^{(b)}. \tag{10}$$

We write $\mathrm{Route}_\phi(q_b)$ for this gate–amplitude mapping. The base Transformer remains frozen and only the router parameters ϕ are optimized. For a sequence $x = (x_1, \ldots, x_T)$, let $p_{\theta,\phi}$ denote the predictive distribution induced by the feedback-augmented forward pass, with θ fixed. We train the router with next-token language-model loss and lightweight route regularization:

$$\mathcal{L}(\phi) = \mathbb{E}_x \left[-\frac{1}{T-1} \sum_{t=1}^{T-1} \log p_{\theta,\phi}(x_{t+1} \mid x_{\leq t}) \right] + \frac{\lambda_m}{|\mathcal{C}|} \mathbb{E}_b \|\tilde{m}^{(b)}\|_1 + \frac{\lambda_\alpha}{|\mathcal{C}|} \mathbb{E}_b \|\tilde{\alpha}^{(b)}\|_2^2. \tag{11}$$

The language-model term scores each route by its effect on prediction, while the auxiliary terms favor selective supports and bounded write strengths. Here $\tilde{m}^{(b)}$ and $\tilde{\alpha}^{(b)}$ denote vectors over all candidate edges. We train the router under synchronous execution and reuse the same router for both synchronous and streaming evaluation.

4.3 Execution Schedules

The same write operator admits two execution schedules, corresponding to two ways of reusing deeper computation. The **synchronous** schedule performs retrospective refinement: the model first completes a traversal and then uses each token's deeper computation to take a second look at its earlier representation. The **streaming** schedule instead makes refinement continuous, carrying the deeper interpretation formed by the processed prefix forward to shape the next token's representation.

Synchronous schedule. The first traversal records no-feedback reference states $z^{\mathrm{ref}} = z^{(0)}$ and forms the current-token payload,

$$p_e^{\mathrm{sync}}(t) = p_e(t) = z_{s_e}^{(0)}(t) - z_{d_e}^{(0)}(t). \tag{12}$$

The model then performs a second traversal using $p_e(t) = p_e^{\mathrm{sync}}(t)$.

Table 2: Perplexity on the ten language-modeling corpora (1024-token chunks). Lower is better. We highlight the best and second-best results.

Corpus	Gemma3-1B					Gemma3-4B					Gemma3-12B					Qwen3-4B					Qwen3-8B				
	Base	Orig.	TF-Loop	Stream	Sync.	Base	Orig.	TF-Loop	Stream	Sync.	Base	Orig.	TF-Loop	Stream	Sync.	Base	Orig.	TF-Loop	Stream	Sync.	Base	Orig.	TF-Loop	Stream	Sync.
arXiv	26.7	23.7	26.5	22.9	**22.5**	22.3	19.2	22.0	18.4	**18.0**	36.9	26.3	35.9	24.7	**23.7**	18.3	16.5	18.2	15.7	**15.4**	14.5	12.8	14.3	12.3	**12.1**
BigPatent	11.8	10.7	12.0	10.6	**10.4**	9.7	9.0	9.6	8.7	**8.5**	20.3	15.4	20.0	14.5	**14.0**	7.8	7.3	7.9	7.1	**7.0**	6.6	6.2	6.6	6.0	**5.9**
BillSum	5.2	5.1	5.1	**4.9**	5.0	4.0	3.8	4.0	3.7	**3.6**	4.8	4.3	4.9	4.1	**4.0**	4.7	4.5	4.6	4.6	**4.3**	3.9	3.7	4.0	3.6	**3.5**
BookSum/Book	30.6	25.6	30.8	25.0	**24.4**	29.7	25.0	29.1	23.9	**23.3**	78.2	53.8	75.3	50.0	**47.2**	17.9	16.3	17.7	15.8	**15.5**	12.4	11.4	12.2	10.9	**10.8**
C4/WebTextLike	25.4	24.1	25.2	23.5	**23.3**	23.8	22.0	23.5	21.2	**20.9**	40.6	37.8	40.1	35.4	**34.8**	21.6	19.9	21.8	19.0	**18.7**	16.3	15.6	16.2	14.7	**14.5**
GovReport	11.9	10.9	12.2	10.6	**10.4**	10.9	9.9	11.0	9.4	**9.3**	28.1	19.5	28.5	18.5	**17.5**	9.9	9.5	9.8	9.0	**8.9**	7.9	7.6	8.0	7.3	**7.2**
LAMBADA	36.7	35.9	36.1	36.0	**34.8**	28.2	26.5	27.7	25.4	**25.1**	25.8	24.8	25.3	23.6	**23.3**	35.5	31.9	34.9	30.5	**30.0**	27.8	25.4	27.4	24.7	**24.3**
Newsroom	14.3	13.9	14.5	13.6	**13.4**	11.7	11.2	11.9	**10.6**	10.7	13.7	12.4	14.0	11.7	**11.4**	20.6	18.7	20.9	18.0	**17.7**	15.2	13.9	15.3	13.5	**13.3**
PG19	22.4	19.0	22.2	18.3	**17.8**	19.5	16.4	19.3	15.7	**15.3**	53.1	35.4	51.5	33.3	**31.3**	25.8	22.8	25.6	21.7	**21.2**	19.0	17.3	18.9	16.3	**16.1**
PubMed	16.2	14.4	16.3	14.0	**13.8**	11.8	10.7	11.6	10.3	**10.1**	26.1	19.6	25.6	18.3	**17.5**	11.6	10.8	11.5	10.5	**10.4**	9.1	8.7	9.0	**8.3**	8.4
Mean	20.1	18.3	20.1	17.9	**17.6**	17.2	15.4	17.0	14.7	**14.5**	32.8	24.9	32.1	23.4	**22.5**	17.4	15.8	17.3	15.2	**14.9**	13.3	12.3	13.2	11.8	**11.6**

Table 3: Accuracy on eight reasoning benchmarks. Each model group reports the base model, Original Recirculation, Training-Free Looped, ReFlux-streaming, and ReFlux-synchronous. Higher is better. We highlight the best and second-best results.

Benchmark	Gemma3-1B					Gemma3-4B					Qwen3-8B				
	Base	Orig.	TF-Loop	Stream	Synchronous	Base	Orig.	TF-Loop	Stream	Synchronous	Base	Orig.	TF-Loop	Stream	Synchronous
ARC-Easy	73.2	73.4$_{\uparrow 0.2}$	74.5$_{\uparrow 1.3}$	**74.8**$_{\uparrow 1.6}$	74.7$_{\uparrow 1.5}$	82.9	83.0$_{\uparrow 0.1}$	83.7$_{\uparrow 0.8}$	84.6$_{\uparrow 1.7}$	**84.9**$_{\uparrow 2.0}$	80.9	81.2$_{\uparrow 0.3}$	81.6$_{\uparrow 0.7}$	82.1$_{\uparrow 1.2}$	**82.5**$_{\uparrow 1.6}$
ARC-Challenge	35.7	35.8$_{\uparrow 0.1}$	36.5$_{\uparrow 0.8}$	37.4$_{\uparrow 1.7}$	**37.8**$_{\uparrow 2.1}$	56.2	56.4$_{\uparrow 0.2}$	57.5$_{\uparrow 1.3}$	58.0$_{\uparrow 1.8}$	**58.4**$_{\uparrow 2.2}$	56.7	56.7$_{\uparrow 0.0}$	57.4$_{\uparrow 0.7}$	58.0$_{\uparrow 1.3}$	**58.5**$_{\uparrow 1.8}$
MMLU	26.2	26.3$_{\uparrow 0.1}$	**27.8**$_{\uparrow 1.6}$	27.5$_{\uparrow 1.3}$	27.4$_{\uparrow 1.2}$	59.5	59.9$_{\uparrow 0.4}$	60.7$_{\uparrow 1.2}$	60.6$_{\uparrow 1.1}$	**60.8**$_{\uparrow 1.3}$	72.9	73.1$_{\uparrow 0.2}$	**74.3**$_{\uparrow 1.4}$	74.1$_{\uparrow 1.2}$	73.9$_{\uparrow 1.0}$
HellaSwag	62.1	62.0$_{\downarrow 0.1}$	62.0$_{\downarrow 0.1}$	**62.6**$_{\uparrow 0.5}$	62.5$_{\uparrow 0.4}$	77.2	77.0$_{\downarrow 0.2}$	77.0$_{\downarrow 0.2}$	77.6$_{\uparrow 0.4}$	**77.9**$_{\uparrow 0.7}$	74.9	74.8$_{\downarrow 0.1}$	75.0$_{\uparrow 0.1}$	75.3$_{\uparrow 0.4}$	**75.5**$_{\uparrow 0.6}$
Winogrande	58.3	58.2$_{\downarrow 0.1}$	58.4$_{\uparrow 0.1}$	58.6$_{\uparrow 0.3}$	**58.7**$_{\uparrow 0.4}$	64.4	64.4$_{\uparrow 0.0}$	64.6$_{\uparrow 0.2}$	**64.9**$_{\uparrow 0.5}$	64.8$_{\uparrow 0.4}$	67.7	67.8$_{\uparrow 0.1}$	67.4$_{\downarrow 0.3}$	68.0$_{\uparrow 0.3}$	**68.2**$_{\uparrow 0.5}$
GSM8K	1.7	1.9$_{\uparrow 0.2}$	2.1$_{\uparrow 0.4}$	3.7$_{\uparrow 2.0}$	**4.9**$_{\uparrow 3.2}$	38.4	38.5$_{\uparrow 0.1}$	39.1$_{\uparrow 0.7}$	41.2$_{\uparrow 2.8}$	**41.5**$_{\uparrow 3.1}$	88.6	88.9$_{\uparrow 0.3}$	88.8$_{\uparrow 0.2}$	90.1$_{\uparrow 1.5}$	**90.4**$_{\uparrow 1.8}$
HotpotQA	42.3	42.4$_{\uparrow 0.1}$	42.7$_{\uparrow 0.4}$	45.5$_{\uparrow 3.2}$	**47.0**$_{\uparrow 4.7}$	49.4	49.6$_{\uparrow 0.2}$	49.2$_{\downarrow 0.2}$	53.3$_{\uparrow 3.9}$	**53.5**$_{\uparrow 4.1}$	57.2	57.3$_{\uparrow 0.1}$	57.6$_{\uparrow 0.4}$	61.4$_{\uparrow 4.2}$	**61.7**$_{\uparrow 4.5}$
2WikiMultiHopQA	24.6	24.7$_{\uparrow 0.1}$	24.9$_{\uparrow 0.3}$	28.3$_{\uparrow 3.7}$	**29.7**$_{\uparrow 5.1}$	38.5	38.6$_{\uparrow 0.1}$	38.9$_{\uparrow 0.4}$	42.8$_{\uparrow 4.3}$	**43.2**$_{\uparrow 4.7}$	44.5	44.7$_{\uparrow 0.2}$	45.0$_{\uparrow 0.5}$	49.1$_{\uparrow 4.6}$	**49.5**$_{\uparrow 5.0}$
Mean	**40.5**	**40.6**$_{\uparrow 0.1}$	**41.1**$_{\uparrow 0.6}$	**42.3**$_{\uparrow 1.8}$	**42.8**$_{\uparrow 2.3}$	**58.3**	**58.4**$_{\uparrow 0.1}$	**58.8**$_{\uparrow 0.5}$	**60.4**$_{\uparrow 2.1}$	**60.6**$_{\uparrow 2.3}$	**67.9**	**68.1**$_{\uparrow 0.2}$	**68.4**$_{\uparrow 0.5}$	**69.8**$_{\uparrow 1.9}$	**70.0**$_{\uparrow 2.1}$

Streaming schedule. The streaming schedule carries a lightweight state across token positions. Let $\hat{z}_\ell(t)$ denote the states recorded during its single, feedback-augmented traversal. At position t, the write uses only the cached payload from the preceding position,

$$p_e^{\text{stream}}(t) = P_e(t-1), \qquad P_e(0) = 0, \tag{13}$$

while the traversal records current states and updates the cache for the next position with $P_e(t) \leftarrow \hat{z}_{s_e}(t) - \hat{z}_{d_e}(t)$. Algorithm 1 in Appendix H summarizes route selection and both execution modes.

5 Experiments

In this section, we conduct extensive experiments to answer: Does ReFlux improve performance across models (Sec 5.2)? How do its gains trade off against computational cost (Sec 5.3)? What mechanisms underlie its effectiveness (Sec 5.4)?

5.1 Experiment Setup

Models and Data. We evaluate ReFlux on five models: Gemma3-1B, Gemma3-4B, Gemma3-12B, Qwen3-4B, and Qwen3-8B (Team et al., 2025; Yang et al., 2025). The router is trained on a compact calibration mixture from the training splits of C4 (Raffel et al., 2020), arXiv (Gao et al., 2020), and PG19 (Rae et al., 2019), covering general web text, scientific writing, and long-form documents. We use 4,096 1,024-token windows from each calibration corpus for router training. For language modeling, we use ten corpora: arXiv (Gao et al., 2020), BigPatent (Sharma et al., 2019), BillSum (Kornilova & Eidelman, 2019), BookSum (Kryściński et al., 2022), C4 (Raffel et al., 2020), GovReport (Huang et al., 2021), LAMBADA (Paperno et al., 2016), Newsroom (Grusky et al., 2018), PG19 (Rae et al., 2019), and PubMed (Gao et al., 2020). We use the validation split for C4 and PG19 and the test split for the remaining corpora. We evaluate reasoning with eight standard benchmarks: MMLU (Hendrycks et al., 2020), ARC-Easy, ARC-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), Winogrande (Sakaguchi et al., 2021), GSM8K (Cobbe et al., 2021), HotpotQA (Yang et al., 2018), and 2WikiMultiHopQA (Ho et al., 2020). HotpotQA and 2WikiMultiHopQA provide complementary multi-hop settings in which the required evidence is distributed across passages, while the other benchmarks measure knowledge, science, and reasoning under shorter contexts.

Baselines. We compare against the base model, the original Recirculation (Mozer et al., 2026), and Training-Free Looped Transformers (TF-Loop) (Chen et al., 2026). TF-Loop applies a contiguous intermediate layer window to the model with damped refinement.

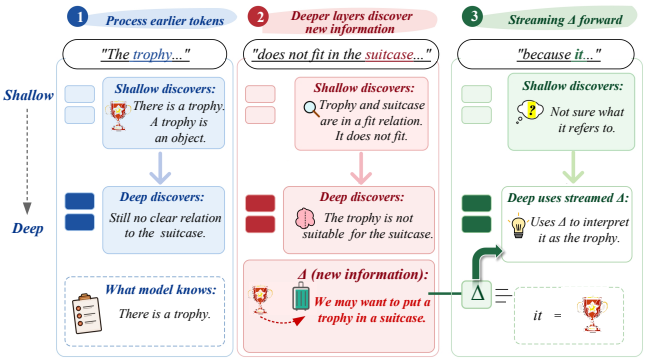


Figure 5: Deep computation as a predictive signal. In *"The trophy does not fit in the suitcase because it is too big,"* shallow layers may encode only coarse semantics when processing *"it"*, while previous deeper computation can establish the relation between the *trophy* and *suitcase*. Instead of revisiting the same token to apply this refinement, ReFlux-streaming carries the Δ forward and makes it available to subsequent tokens, allowing them to benefit from computation already performed.

Table 4: Decoding efficiency and C4 language-modeling quality. Efficiency metrics are color-coded.

	GEMMA3-1B					QWEN3-4B				
Method	**Runtime(×)**	**FLOPs(×)**	**Memory(GB)**	**Latency(ms/tok)**	**PPL**	**Runtime(×)**	**FLOPs(×)**	**Memory(GB)**	**Latency(ms/tok)**	**PPL**
Base	1.00	1.00	1.88	15.05	25.4	1.00	1.00	7.56	16.05	21.6
Orig.	1.88	2.00	1.90	28.26	24.1	1.94	2.00	7.58	31.14	19.9
TF-Loop	1.34	1.31	1.88	20.16	25.2	1.21	1.22	7.56	19.42	21.8
Streaming	1.02	1.00	1.88	15.34	23.5	1.08	1.00	7.57	17.35	19.0
Synchronous	1.91	2.00	1.89	28.71	23.3	1.97	2.00	7.64	31.59	18.7

Implementation Details. We partition each input into routing blocks and make one causal routing decision per block. The router consists of a two-layer MLP with hidden width 512 and GELU nonlinearity, followed by separate linear heads for edge gates and write amplitudes. We set $\xi = 0.6$, $\lambda_m = 10^{-3}$ and $\lambda_\alpha = 10^{-2}$. Full details are given in App. E.

5.2 Main Results

Tables 2 and 3 show that ReFlux improves both language modeling and reasoning across model families. On the ten-corpus PPL suite, ReFlux consistently outperforms both the Base model and Original Recirculation across all corpora. The improvement remains consistent across the Gemma3 and Qwen3 families and across different model scales. Original Recirculation also helps but remains consistently weaker, while TF-Loop often stays close to the base model. On downstream benchmarks, ReFlux improves average accuracy from 40.5 to 42.3/42.8 on Gemma3-1B, from 58.3 to 60.4/60.6 on Gemma3-4B, and from 67.9 to 69.8/70.0 on Qwen3-8B. The gains are largest on HotpotQA and 2WikiMultiHopQA, where answers require combining evidence across passages. We also compare ReFlux with CoT (Wei et al., 2022) and their combination in App. D. Together, the results suggest that the benefit is not merely re-executing depth, but accessing the computation that was newly produced.

5.3 Efficiency Analysis

Table 4 compares these gains against inference cost. ReFlux-streaming keeps the backbone at $1\times$ FLOPs, adds almost no memory, and runs at only $1.02\times/1.08\times$ wall-clock cost during decoding, yet its PPL remains close to synchronous execution. As demonstrated in Fig. 5, useful feedback need not always revisit the same token: the newly accumulated Δ can be carried forward and bias the interpretation of subsequent tokens, serving as a predictive signal. Synchronous ReFlux gives the strongest PPL by performing a stricter second traversal, but its roughly $1.9\times$ runtime matches the cost profile of Original Recirculation. Details are given in App. E.2, with a Pareto view in Fig. 9.

5.4 Understanding ReFlux

Routing Behavior. We ask whether the router learns a genuinely input-conditioned policy. For each block, Fig.6 shows three consistent routing patterns: higher-loss context leads the subsequent route to activate more edges, frequently used source-target pairs form localized cross-depth hotspots, and routes are more stable within task families than across families. These results suggest that the router learns a structured, input-conditioned computation policy. Details are given in Appendix E.5.

Ablations. Table 5 summarizes the ablations, with the full protocol in App. F.1. Replacing the increment with the full source state degrades performance, showing that the depth increment is a more effective feedback payload. Removing either strength adaptation or input-conditioned support also causes consistent degradation, indicating that both write strength and route selection contribute. In contrast, Heuristic-Delta performs substantially worse and is the only variant to fall below the base model, suggesting that static representation geometry alone is insufficient to determine which increments are useful. We compare ReFlux with budget-matched trainable controls in App. F.2.

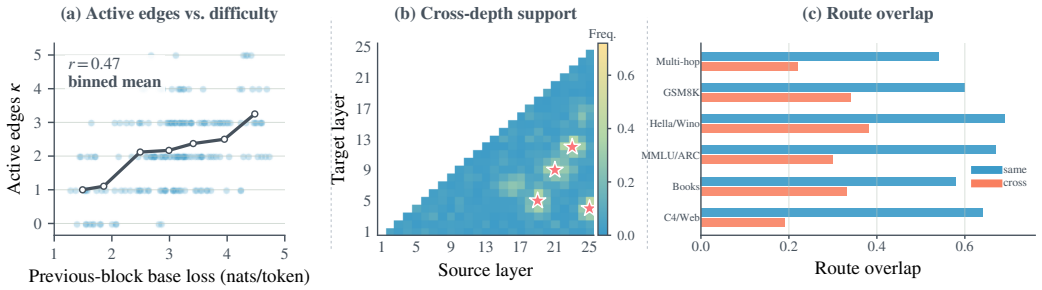


Figure 6: Routing behavior summary. (a) Integer active-edge count for routing block $b{+}1$ against the base-model NLL of the preceding block b; the line connects binned means. (b) Source-target heatmap of aggregated activation frequencies, with the most frequently used pairs marked by stars. (c) Comparison of same-family and cross-family route overlap.

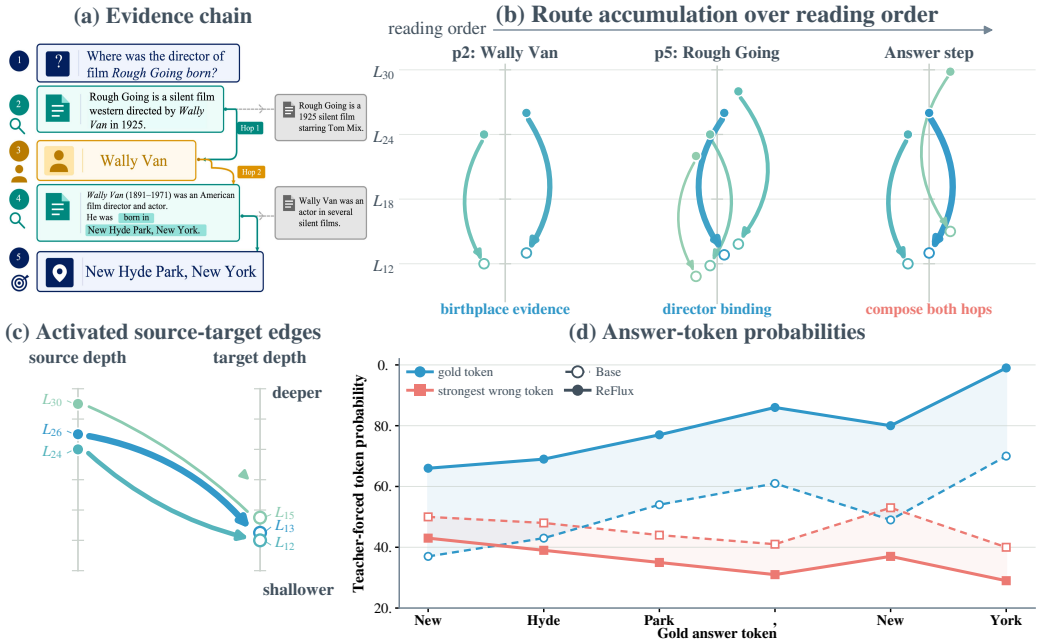


Figure 7: Case study (App. G). (a) The question's evidence chain, (b) how route support accumulates (we set the block size to 1 token for visualization), (c) the final active source-target edges, and (d) teacher-forced token probabilities for the gold answer path and the strongest wrong competitor.

Case Study. We use one representative 2WikiMultiHopQA example to illustrate how ReFlux supports evidence composition across hops and states (Fig.7). The question requires first binding *Rough Going* to *Wally Van* and then carrying that bridge entity to the birthplace answer. ReFlux concentrates its routed support on the bridge passage, so the final decision reuses the earlier hop rather than rebuilding it from scratch. In the trace, this leads to a larger gap between the gold answer path and the strongest distractor, enabling ReFlux to answer correctly where the base model fails.

Table 5: Ablations on Gemma3-4B. We report C4 PPL and two reasoning accuracies, ARC-E and ARC-C.

Config	C4 PPL ↓	ARC-E ↑	ARC-C ↑
ReFlux-Sync.	**20.9**	**84.9**	**58.4**
Base	23.8$_{\uparrow 2.9}$	82.9$_{\downarrow 2.0}$	56.2$_{\downarrow 2.2}$
Learned-full	22.1$_{\uparrow 1.2}$	83.1$_{\downarrow 1.8}$	55.9$_{\downarrow 2.5}$
Uniform-strength	21.6$_{\uparrow 0.7}$	84.1$_{\downarrow 0.8}$	57.3$_{\downarrow 1.1}$
Fixed-Delta	21.7$_{\uparrow 0.8}$	84.2$_{\downarrow 0.7}$	57.5$_{\downarrow 0.9}$
Heuristic-Delta	24.3$_{\uparrow 3.4}$	82.0$_{\downarrow 2.9}$	55.8$_{\downarrow 2.6}$

6 Conclusion

In this work, we identified the depth increment (Δ) as a precise, composable feedback signal in Transformers and proposed ReFlux, a learnable, input-conditioned feedback graph over frozen models. Extensive experiments across various models demonstrated its effectiveness. Looking forward, integrating this adaptive Δ-routing into pre-training or test-time compute presents exciting directions. More broadly, ReFlux shows that frozen LLMs can recover useful capability by letting later computation reshape earlier interpretation, giving the model a second look with new discoveries.

7 AI USAGE STATEMENT

During the preparation of this manuscript, we used advanced language models solely for editorial assistance. Their use was restricted to language-level tasks, including grammar correction, sentence refinement, and improving the clarity and coherence of the presentation. The models were not used to generate or determine the research ideas, technical methodology, experimental design, or reported findings. All scientific contributions and conclusions presented in this work were developed and verified by the authors.

8 ETHICS STATEMENT

This work does not involve human subjects, personally identifiable information, or the collection of sensitive or private data. All experiments are conducted on publicly available pretrained models and datasets, following their respective usage and licensing terms. The proposed method is intended to study efficient and controllable information propagation in language models and does not introduce any new data collection or human-facing deployment. We are not aware of any specific ethical, privacy, security, or fairness concerns arising from the methodology or experiments presented in this work. We have also disclosed our source code, experimental protocols and evaluation procedures to support transparent and reproducible research.

9 REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the ReFlux architecture, feedback construction, routing mechanism, and inference procedures in Section 4 and Appendix H. The experimental setup, including the evaluated models, datasets, prompts, evaluation protocols, and implementation details, is documented in Section 5.1 and Appendix E. The supplementary material includes the source code, configuration files, and evaluation scripts.

REFERENCES

Alan D. Baddeley. Working memory: looking back and looking forward. *Nature Reviews Neuroscience*, 4:829–839, 2003. URL https://api.semanticscholar.org/CorpusID:3337171.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, et al. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. *Advances in Neural Information Processing Systems*, 38:96572–96617, 2026.

Andrea Banino, Jan Balaguer, and Charles Blundell. Pondernet: Learning to ponder. *arXiv preprint arXiv:2107.05407*, 2021.

Nora Belrose, Igor Ostrovsky, Lev McKinney, Zach Furman, Logan Smith, Danny Halawi, Stella Biderman, and Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens. *arXiv preprint arXiv:2303.08112*, 2023.

Joschka Braun, Carsten Eickhoff, David Krueger, Seyed Ali Bahrainian, and Dmitrii Krasheninnikov. Understanding (un) reliability of steering vectors in language models. *arXiv preprint arXiv:2505.22637*, 2025.

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.

Lizhang Chen, Jonathan Li, Chen Liang, Ni Lao, and Qiang Liu. Training-free looped transformers. *arXiv preprint arXiv:2605.23872*, 2026.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

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.

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

Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. Implicit chain of thought reasoning via knowledge distillation. *arXiv preprint arXiv:2311.01460*, 2023.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, et al. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 1(1):12, 2021.

Harriet Feldman and Karl J Friston. Attention, uncertainty, and free-energy. *Frontiers in human neuroscience*, 4:215, 2010.

D J Felleman and David C. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral cortex*, 1 1:1–47, 1991. URL https://api.semanticscholar.org/CorpusID:9334496.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *Advances in Neural Information Processing Systems*, 38: 41340–41391, 2026.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 5484–5495, 2021.

Mor Geva, Avi Caciularu, Kevin Wang, and Yoav Goldberg. Transformer feed-forward layers build predictions by promoting concepts in the vocabulary space. In *Proceedings of the 2022 conference on empirical methods in natural language processing*, pp. 30–45, 2022.

Asma Ghandeharioun, Avi Caciularu, Adam Pearce, Lucas Dixon, and Mor Geva. Patchscopes: A unifying framework for inspecting hidden representations of language models. *arXiv preprint arXiv:2401.06102*, 2024.

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.

Alex Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Andrey Gromov, Kushal Tirumala, Hassan Shapourian, Paolo Glorioso, and Daniel A Roberts. The unreasonable ineffectiveness of the deeper layers. In *International Conference on Learning Representations*, volume 2025, pp. 81906–81920, 2025.

Max Grusky, Mor Naaman, and Yoav Artzi. Newsroom: A dataset of 1.3 million summaries with diverse extractive strategies. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 708–719, 2018.

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.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Luyang Huang, Shuyang Cao, Nikolaus Parulian, Heng Ji, and Lu Wang. Efficient attentions for long document summarization. In *Proceedings of the 2021 conference of the north American chapter of the association for computational linguistics: Human language technologies*, pp. 1419–1436, 2021.

Xinyan Jiang, Wenjing Yu, Di Wang, and Lijie Hu. Global evolutionary steering: Refining activation steering control via cross-layer consistency. *arXiv preprint arXiv:2603.12298*, 2026.

Alexia Jolicoeur-Martineau. Less is more: Recursive reasoning with tiny networks. *arXiv preprint arXiv:2510.04871*, 2025.

Jeonghoon Kim, Byeongchan Lee, Cheonbok Park, Yeontaek Oh, Beomjun Kim, Taehwan Yoo, Seongjin Shin, Dongyoon Han, Jinwoo Shin, and Kang Min Yoo. Peri-LN: Revisiting normalization layer in the transformer architecture. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 30400–30436. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/kim25u.html.

Harsh Kohli, Srinivasan Parthasarathy, Huan Sun, and Yuekun Yao. Loop, think, & generalize: Implicit reasoning in recurrent-depth transformers. *arXiv preprint arXiv:2604.07822*, 2026.

Yeskendir Koishekenov, Aldo Lipani, and Nicola Cancedda. Encode, think, decode: Scaling test-time reasoning with recursive latent thoughts. *arXiv preprint arXiv:2510.07358*, 2025.

Anastassia Kornilova and Vladimir Eidelman. Billsum: A corpus for automatic summarization of us legislation. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pp. 48–56, 2019.

Wojciech Kryściński, Nazneen Rajani, Divyansh Agarwal, Caiming Xiong, and Dragomir Radev. Booksum: A collection of datasets for long-form narrative summarization. In *Findings of the association for computational linguistics: EMNLP 2022*, pp. 6536–6558, 2022.

Kenneth Li, Oam Patel, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Inference-time intervention: Eliciting truthful answers from a language model. *Advances in neural information processing systems*, 36:41451–41530, 2023.

Pengxiang Li, Lu Yin, and Shiwei Liu. Mix-ln: Unleashing the power of deeper layers by combining pre-ln and post-ln. *arXiv preprint arXiv:2412.13795*, 2024.

Ziyue Li, Yang Li, and Tianyi Zhou. Skip a layer or loop it? test-time depth adaptation of pretrained llms. *arXiv preprint arXiv:2507.07996*, 2025.

Ziyue Li, Yang Li, and Tianyi Zhou. Skip a Layer or Loop It? Learning Program-of-Layers in LLMs. In *Forty-third International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/pdf/2606.06574.

Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20192–20204, 2025.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372, 2022.

William Merrill and Ashish Sabharwal. The expressive power of transformers with chain of thought. In *International Conference on Learning Representations*, volume 2024, pp. 7690–7706, 2024.

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

Kyle O'Brien, David Majercak, Xavier Fernandes, Richard Edgar, Blake Bullwinkel, Jingya Chen, Harsha Nori, Dean Carignan, Eric Horvitz, and Forough Poursabzi-Sangdeh. Steering language model refusal with sparse autoencoders. *arXiv preprint arXiv:2411.11296*, 2024.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc-Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 1525–1534, 2016.

Jacob Pfau, William Merrill, and Samuel R Bowman. Let's think dot by dot: Hidden computation in transformer language models. *arXiv preprint arXiv:2404.15758*, 2024.

Jack W Rae, Anna Potapenko, Siddhant M Jayakumar, and Timothy P Lillicrap. Compressive transformers for long-range sequence modelling. *arXiv preprint arXiv:1911.05507*, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Rajesh PN Rao and Dana H Ballard. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nature neuroscience*, 2(1):79–87, 1999.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

Nina Rimsky, Nick Gabrieli, Julian Schulz, Meg Tong, Evan Hubinger, and Alexander Turner. Steering llama 2 via contrastive activation addition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15504–15522, 2024.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

Eva Sharma, Chen Li, and Lu Wang. Bigpatent: A large-scale dataset for abstractive and coherent summarization. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 2204–2213, 2019.

Richard M Shiffrin and Walter Schneider. Controlled and automatic human information processing: Ii. perceptual learning, automatic attending and a general theory. *Psychological review*, 84(2):127, 1977.

Asa Cooper Stickland, Alexander Lyzhov, Jacob Pfau, Salsabila Mahdi, and Samuel R Bowman. Steering without side effects: Improving post-deployment control of language models. *arXiv preprint arXiv:2406.15518*, 2024.

Mingjie Sun, Xinlei Chen, J Zico Kolter, and Zhuang Liu. Massive activations in large language models. *arXiv preprint arXiv:2402.17762*, 2024.

Wenfang Sun, Xinyuan Song, Pengxiang Li, Lu Yin, Yefeng Zheng, and Shiwei Liu. The curse of depth in large language models. *Advances in Neural Information Processing Systems*, 38: 163104–163136, 2026.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Kimi Team, Guangyu Chen, Yu Zhang, Jianlin Su, Weixin Xu, Siyuan Pan, Yaoyu Wang, Yucheng Wang, Guanduo Chen, Bohong Yin, et al. Attention residuals. *arXiv preprint arXiv:2603.15031*, 2026.

Alexander Matt Turner, Lisa Thiergart, Gavin Leech, David Udell, Juan J Vazquez, Ulisse Mini, and Monte MacDiarmid. Steering language models with activation engineering. *arXiv preprint arXiv:2308.10248*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Shaowen Wang, Ge Zhang, Kairong Luo, Yuhao Wu, Shaofan Liu, Jiaheng Liu, Wenhao Huang, Shen Yan, and Jian Li. Smelt: Scaling laws for compute-matched moe looped transformers. *arXiv preprint arXiv:2609.01343*, 2026a.

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

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.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International conference on machine learning*, pp. 10524–10533. PMLR, 2020.

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

Liu Yang, Kangwook Lee, Robert Nowak, and Dimitris Papailiopoulos. Looped transformers are better at learning learning algorithms. In *International conference on learning representations*, volume 2024, pp. 42195–42214, 2024a.

Yifei Yang, Zouying Cao, and Hai Zhao. Laco: Large language model pruning via layer collapse. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 6401–6417, 2024b.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Wencheng Ye, Xiaoyang Yuan, Yi Bin, Hengyu Jin, Liang Peng, Pengpeng Zeng, and Heng Tao Shen. Riser: Orchestrating latent reasoning skills for adaptive activation steering. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 4627–4644, 2026.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 4791–4800, 2019.

Boyi Zeng, Shixiang Song, Siyuan Huang, Yixuan Wang, He Li, Ziwei He, Xinbing Wang, Zhouhan Lin, et al. Ponderlm: Pretraining language models to ponder in continuous space. In *International Conference on Learning Representations*, volume 2026, pp. 91874–91893, 2026.

Xuanming Zhang, Sining Zhoubian, Yuxuan Chen, Tianyi Tang, An Yang, Sean Du, Chujie Zheng, Fei Huang, Dayiheng Liu, Gao Huang, and Jingren Zhou. Deeper is not always better: Mitigating the alignment tax via confident layer decoding, 2026. URL https://arxiv.org/abs/2606.21906.

Table 6: Positioning of ReFlux among depth-feedback and activation-editing methods. ✓ / ✗: property holds / does not hold; ∼: partially. †requires pretraining or continued training. The single-traversal column for ReFlux refers to its streaming schedule. Sources: UT (Dehghani et al., 2018); Looped LM (Yang et al., 2024a); SMELT (Wang et al., 2026a); MoR (Bae et al., 2026); MoDA (Zhu et al., 2026); T^{2}MLR (Cai et al., 2026); TF-Loop (Chen et al., 2026); CoLa (Li et al., 2025); CAA (Rimsky et al., 2024); ITI (Li et al., 2023); Patchscopes (Ghandeharioun et al., 2024); Recirculation (Mozer et al., 2026).

Method	adds effective depth	no backbone retraining	any frozen checkpoint	single traversal	in-place write	endogenous payload	multi-edge feedback
UT / Looped LM / SMELT†	✓	✗	✗	✗	✓	✓	✗
MoR†	✓	✗	✗	∼	✓	✓	✗
MoDA†	∼	✗	✗	✓	✓	✓	∼
T^{2}MLR†	✓	✗	✗	✓	✓	✓	✗
TF-Loop	✓	✓	✓	✗	✓	✓	✗
CoLa	✓	✓	✓	✗	✓	✓	∼
CAA / ITI	✗	✓	✓	✓	✓	✗	✗
Patchscopes	✗	✓	✓	✗	✗	✓	✗
Recirculation	✓	✓	✓	✗	✓	✓	✗
ReFlux (ours)	✓	✓	✓	✓	✓	✓	✓

Lianghui Zhu, Yuxin Fang, Bencheng Liao, Shijie Wang, Tianheng Cheng, Zilong Huang, Chen Chen, Lai Wei, Yutao Zeng, Ya Wang, Yi Lin, Yu Li, and Xinggang Wang. Mixture-of-depths attention, 2026. URL https://arxiv.org/abs/2603.15619.

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 Extended Related Works

A.1 Residual Streams

The residual stream provides the basic substrate for the depth-wise information flow considered by ReFlux. Originating from the residual connections of ResNets (He et al., 2016), this additive structure preserves an identity path while accumulating learned updates across layers. In Transformers, layers read from and write to a shared residual stream, which can therefore be viewed as a communication channel between components (Elhage et al., 2021). Normalization further shapes how these updates accumulate: pre-norm architectures improve optimization stability in deep Transformers (Xiong et al., 2020), while subsequent work identifies reduced contributions from later blocks and explores alternatives such as Peri-LN and Mix-LN (Kim et al., 2025; Li et al., 2024). Recent work has begun to treat this stream not merely as a sequence of overwritten states, but as a structured record of computation across depth. Massive activations and attention sinks show that information flow can become concentrated on particular tokens (Sun et al., 2024), while Attention Residuals (Team et al., 2026) demonstrates that contributions from different depths can be selectively aggregated. MoDA (Zhu et al., 2026) takes this idea further by giving attention heads explicit access to depth-specific keys and values produced by shallower layers at the same token position. These works suggest that information accumulated along depth can remain individually accessible rather than being represented only through the current full residual state. However, most related designs propagate information forward across depth, from shallower to deeper layers. Two exceptions reverse this direction. T^{2}MLR (Cai et al., 2026) trains a gated fusion pathway along time, fusing a cached middle-layer state of the previous token into an early layer of the current token, while Recirculation (Mozer et al., 2026) writes full deep states back into shallower streams during inference. ReFlux combines the deep-to-shallow direction of these designs with an increment payload and a frozen backbone: it feeds $\Delta_{d\to s} = z_s - z_d$, the computation newly accumulated between two depths, back into shallower layers of the same token or, through streaming, of subsequent tokens.

A.2 Depth Utilization and Redundancy

A growing body of evidence suggests that depth is used unevenly across layers. ShortGPT (Men et al., 2025) finds that many mid-depth layers can be removed with limited performance loss, while Gromov et al. (2025) show that some models tolerate removing up to half of their deeper layers; LaCo

(Yang et al., 2024b) similarly collapses rear layers into earlier ones with little quality loss. Recent analysis attributes part of this redundancy to normalization: LayerNorm and RMSNorm make block outputs relatively insensitive to input magnitude, causing deeper layers to contribute progressively less (Sun et al., 2026). Yet reduced contribution does not mean that depth stops computing useful information. Intermediate layers tend to make incremental, directionally faithful updates, whereas later layers can shift predictions toward generic, alignment-preferred tokens, thus decoding from an intermediate entropy-valley layer can outperform decoding from the final layer (Zhang et al., 2026). At the execution-path level, POLAR shows that input-specific programs can skip or repeat pretrained layer segments to outperform the standard forward pass (Li et al., 2026). Meanwhile, feed-forward layers act as key–value memories that progressively build predictions by promoting concepts in vocabulary space (Geva et al., 2021; 2022). Causal tracing further localizes factual recall to mid-layer MLPs, where editing a single module can rewrite factual associations (Meng et al., 2022), while affine probes recover increasingly sharp predictions from intermediate states (Belrose et al., 2023). The picture that emerges is not that deeper computation is redundant, but that its output can be partly redundant even when the computation itself is informative. ReFlux builds on this distinction by propagating the newly computed increment rather than replaying the full state.

A.3 Cognitive Principles

Our computation reuse perspective also resonates with a broader view of information processing as hierarchical, recurrent, and selective. Predictive coding proposes that higher cortical areas send predictions to lower areas to refine the interpretation of incoming activity (Rao & Ballard, 1999), while the cortical hierarchy is organized through reciprocal connections across levels (Felleman & Essen, 1991). Cognitive theories further suggest that limited-capacity working memory selectively maintains task-relevant information (Baddeley, 2003), with processing shaped by the interaction between controlled and automatic mechanisms (Shiffrin & Schneider, 1977). Related theories of attention and free energy also emphasize that the precision assigned to a signal can depend on its content (Feldman & Friston, 2010). These principles provide a conceptual interpretation of several design choices in ReFlux: hierarchical feedback, selective routing under capacity constraints, content-dependent modulation, and temporally delayed feedback. In particular, ReFlux propagates newly computed depth increments rather than full representations, and its streaming formulation incorporates these corrections into the ongoing feedforward computation, providing a computational analogue of recurrent information exchange in hierarchical processing.

B Diagnosing and Repairing Cross-Model Failure

The absence of a clear gain from Recirculation on Qwen-Family models has been reported in prior work. Here we try to explain why the same feedback idea transfers unevenly across normalization architectures and to test whether the limitation can be removed. The results below show that the Qwen3 behavior is structured rather than incidental: the outcome depends jointly on the write strength, the target depth, and the representation written back.

B.1 A normalization-dependent failure mode

Let $n_\ell(t) := \|z_\ell(t)\|_2$ denote the residual-stream norm trajectory and let $r_\ell(t) := \|F_\ell(z_\ell, t)\|_2/(n_\ell(t) + \varepsilon)$ denote the relative branch budget at depth ℓ. In a Pre-LN model, both quantities are calibrated during training; a Peri-LN model further normalizes branch outputs before they enter the stream:

$$z^{\mathrm{Peri}}_{\ell+1}(t) = z_\ell(t) + \mathrm{LN}_{\mathrm{out}}\big(F_\ell(\mathrm{LN}_{\mathrm{in}}(z_\ell(t)))\big), \qquad z^{\mathrm{Pre}}_{\ell+1}(t) = z_\ell(t) + F_\ell(\mathrm{LN}(z_\ell(t))).$$

Applying a stream-scale feedback write to Qwen3 can therefore introduce a norm shock

$$\sigma_d(t) := \frac{\|\bar{z}_d(t)\|_2}{\|z_d(t)\|_2 + \varepsilon}, \qquad \bar{z}_d(t) = z_d(t) + u_d(t),$$

where $u_d(t)$ is the injected feedback vector. When $\sigma_d(t)$ departs substantially from one, the write moves the stream off the calibrated trajectory used by the downstream stack. This does not merely rescale the representation: the perturbed stream is still processed by layers whose updates were trained under the original norm profile, so the shock can persist as a directional error rather than being washed out. The cross-depth profiles in App. C are consistent with this picture: the residual state is dominated by a low-dimensional direction that rotates only gradually with depth, so a scale disturbance survives through the remaining blocks.

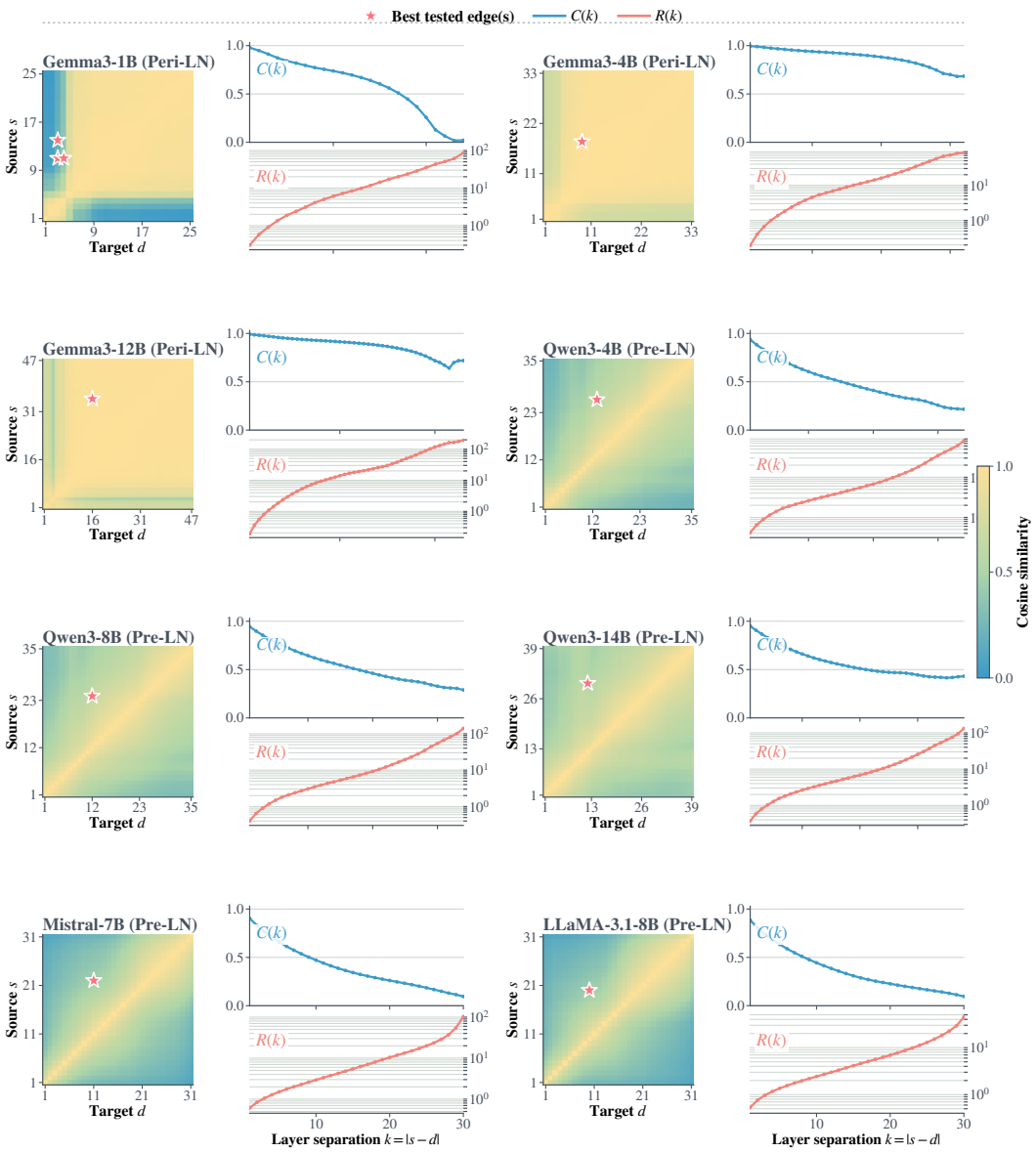


Figure 8: Cross-depth residual-stream geometry across the eight profiled models. Each model panel shows pairwise cosine similarity, cosine similarity $C(k)$, and norm-relative displacement $R(k)$. Stars denote the best tested source-target edge(s) for each model.

B.2 Our repair

We retain the increment as the feedback content, but preserve the norm of the destination stream after writing it. If $\bar{z}_d$ denotes the stream after the payload has been added, we use

$$\widetilde{z}_d(t) = \frac{\|z_d(t)\|_2}{\|\bar{z}_d(t)\|_2 + \varepsilon} \bar{z}_d(t). \tag{14}$$

The normalization protocols used in the experiments are architecture-specific. For Gemma3 (Peri-LN), we use the stream-scale payload normalization $\mathcal{N}_{z_d}$ in Eq. 5 without an additional post-write projection. For Qwen3 (Pre-LN), we use the same target-scale payload normalization followed by the norm-preserving projection in Eq. 14. Thus the two models receive the same increment-based feedback, but under different residual-stream norm protocols.

B.3 Cross-architecture results

The repaired operator turns the previously marginal Qwen3 transfer into a consistent positive result. On Qwen3-4B, it improves C4 perplexity by $+5.99\%$ at 512 tokens and $+6.15\%$ at 1,024 tokens.

Table 7: Positioning of CoT, filler-token, looped, and increment-feedback methods. Filler tokens must be trained to be exploited (Pfau et al., 2024); the router calibration of ReFlux is a single lightweight pass. The last column indicates which of the two bottlenecks each method primarily addresses. ✓ / ✗: property holds / does not hold; ∼: partially. Sources: CoT (Wei et al., 2022); Filler tokens (Pfau et al., 2024); Looped (Dehghani et al., 2018; Yang et al., 2024a).

Method	adds serial steps	extra training	extra token generation	plug-and-play	bottleneck addressed
CoT	✓	✗	5–50×	✓	computation
Filler tokens	✓	✓	$N\times$	✗	computation
Looped	✓	✓	0	✗	computation
ReFlux	✗	∼	0	✓	**access**

In conclusion, these results turn a previously reported negative transfer into a more precise architectural account. The limitation is not that deep-to-shallow feedback is intrinsically ineffective on Pre-LN models, but that an otherwise informative write can violate the residual-stream scale under which the downstream computation was trained. Preserving the destination norm removes this incompatibility while leaving the feedback content itself unchanged: the useful signal remains the depth increment Δ, and the normalization protocol determines whether that signal can be safely received. This distinction separates the semantic design of feedback from its architecture-dependent execution conditions, and provides the basis for transferring the method beyond the models studied here through lightweight stream profiling.

C Cross-depth Geometry Across Models

We apply the same residual-stream profiling protocol across eight models from different families on C4. Fig.8 provides a compact cross-model view: each panel contains the pairwise cosine heatmap and the corresponding similarity and displacement profiles by layer separation. Stars mark the best tested source-target edge for each model.

D Comparison with Chain-of-Thought Reasoning

Since chain-of-thought (CoT) prompting is a standard way of allocating additional computation on reasoning benchmarks, we further compare ReFlux with CoT from the perspective of how additional computation is introduced and used.

D.1 Theoretical Comparison

In CoT, each generated token provides an additional serial reasoning step, allowing the model to construct new intermediate states over a longer computation trajectory. Merrill & Sabharwal (2024) formalize this view by characterizing the expressive power of decoder models as a function of their intermediate computation budget. The role of additional computation is also highlighted by the filler-token setting of Pfau et al. (2024), where meaningless tokens can sometimes match the performance of CoT on controlled tasks, which suggests that part of the benefit of CoT can arise from the additional *computation* itself, rather than the semantic content of its intermediate reasoning steps.

There is an architectural reason why this computation must be purchased in tokens. In a standard decoder-only model, a deep representation computed at one position can influence later computation only through two channels: it is either re-derived by traversing the same depth again and retrieved indirectly through attention, or compressed into the discrete token that carries the context forward through the unembed–decode–embed interface (Cai et al., 2026). Chain-of-thought prompting is best understood as a deliberate exploitation of the second channel: each reasoning step purchases persistence for intermediate computation by paying one generated token, so the state that survives a step boundary is a quantized, low-bandwidth projection of the richer computation that produced it.

ReFlux introduces a latent feedback channel that preserves intermediate representations without routing them through either additional depth or the discrete token interface. Rather than increasing the number of serial reasoning steps, it keeps the depth available to each token fixed while changing how previously computed representations are reused. In the streaming setting, each token still undergoes a single depth-L forward pass: an increment computed at an earlier position is written into a shallow key/value entry and can later be retrieved directly, without re-executing the layers that produced it or passing through the token interface. The synchronous schedule realizes the same feedback through

Table 8: **Four-setting comparison on Qwen3-8B.** We report accuracy and mean generated tokens per example.

	2WikiMultiHopQA		MMLU		GSM8K	
Setting	**Acc**	**Tokens**	**Acc**	**Tokens**	**Acc**	**Tokens**
Base	44.5	97	70.2	64	88.6	189
+ CoT	44.7	326	71.4	508	91.8	1002
ReFlux (sync)	49.5	120	71.2	129	90.4	263
+ CoT	49.9	294	71.1	480	92.0	691

an additional traversal at a fixed $2\times$ FLOPs cost—a cost that, unlike the token-level computation introduced by chain-of-thought, does not grow with the length of the reasoning chain.

This distinction separates two different computational bottlenecks. When a useful intermediate result has not yet been computed, additional serial computation can create it, as in CoT. When the relevant result has already been computed but is inaccessible at the depth where it is needed, however, additional computation is not necessarily the solution. The bottleneck is then one of *access*: ReFlux makes existing computation available to the computation that needs it, rather than repeatedly recomputing it through a longer serial trajectory. Table 7 summarizes this positioning against CoT, filler-token, and looped approaches.

D.2 Empirical Comparison

We evaluate this analysis on three benchmarks spanning the computation–access axis: 2WikiMultiHopQA (composition, access-bound), MMLU (knowledge, neutral), and GSM8K (arithmetic, computation-bound). On Qwen3-8B, we compare four settings under a unified, fully generative protocol: direct answering (Base), zero-shot chain-of-thought prompting (CoT), ReFlux feedback, and their combination (CoT + ReFlux). In every setting, the model generates the final answer from scratch; in the CoT setting, it first generates a detailed, step-by-step reasoning trace before producing the answer. We then extract the answer from the generated text and report both accuracy and the mean number of generated tokens per example. Unlike the likelihood-based multiple-choice evaluation used in the main text, this protocol requires every setting to generate an answer, making the four settings directly comparable in both performance and generation cost.

The four settings form a 2×2 factorial design, $\{\text{CoT}, \text{no-CoT}\} \times \{\text{ReFlux}, \text{no-ReFlux}\}$, allowing us to isolate the contributions of the two bottlenecks identified by our analysis (Table 8). On 2WikiMultiHopQA, ReFlux improves accuracy from 44.5 to 49.5, whereas CoT has almost no effect ($44.5 \rightarrow 44.7$). On GSM8K, the pattern is reversed: CoT provides the larger gain ($88.6 \rightarrow 91.8$), while ReFlux still improves performance to 90.4. MMLU, which lies between these two regimes, shows smaller and more balanced effects from the two interventions. These results are consistent with the proposed bottleneck view: CoT primarily supplies additional serial computation, whereas ReFlux primarily makes previously computed information accessible across depth.

The computational footprints are also markedly asymmetric. CoT substantially extends the generated reasoning trace, adding 229, 444, and 813 tokens on 2WikiMultiHopQA, MMLU, and GSM8K, respectively. Thus, in this fully generative setting, the two interventions differ not only in where they act but also in how their additional computation is exposed at the output level: CoT allocates computation through a longer sequence of generated intermediate tokens, while ReFlux reuses internal computation without requiring additional reasoning tokens.

More interestingly, adding ReFlux to CoT often *shortens* the reasoning trace rather than simply improving its accuracy. The mean generated length drops from 326 to 294 tokens on 2WikiMultiHopQA, from 508 to 480 on MMLU, and from 1002 to 691 on GSM8K, while accuracy is maintained or improved in all three cases. This shortening is consistent with a sub-additive interaction between the two interventions: once previously computed information becomes accessible through ReFlux, part of the serial reasoning that CoT would otherwise need to externalize can be replaced by internal feedback. In other words, ReFlux does not merely add another source of computation on top of CoT; it can change how much explicit serial computation is needed downstream. The two mechanisms therefore remain complementary, but their effects need not be additive: making existing computation accessible can reduce the amount of additional computation that must be generated explicitly.

E Experimental Details

E.1 Router Training Hyperparameters

E.1.1 Architecture

For routing block b, we take the residual states after the final Transformer block over the preceding token block, apply the backbone's layer-normalization convention, and mean-pool over positions to obtain $q_b \in \mathbb{R}^D$. For the first routing block, where no preceding block exists, we use a zero vector and therefore start from the router's learned biases. In synchronous training and evaluation, q_b is computed from the no-feedback reference traversal; in streaming evaluation, it is computed from the feedback-augmented trajectory available at the end of the preceding block. In parallel-prefill generation, the first decode block uses the summary of the final prompt block. The router applies two fully connected layers of width 512, each followed by GELU, with dropout 0.1 between the layers. Two independent linear heads map the resulting representation to $\eta^{(b)}, \rho^{(b)} \in \mathbb{R}^{|\mathcal{C}|}$, producing one gate logit and one amplitude logit per candidate edge. Amplitudes use $\mathrm{softplus}$ and are clipped at 0.25 before the architecture-specific write protocol is applied. The backbone parameters are frozen; gradients update only the router. For a backbone with L Transformer blocks, the candidate set contains every deep-to-shallow pair of block outputs,

$$\mathcal{C} = \{(s, d) : 1 \le d < s \le L\}, \qquad |\mathcal{C}| = \frac{L(L-1)}{2}.$$

This gives 325, 561, and 1,128 candidate edges for Gemma3-1B, Gemma3-4B, and Gemma3-12B, respectively, and 630 edges for both Qwen3-4B and Qwen3-8B. The router therefore learns the useful depth intervals rather than inheriting a geometry-based pruning rule.

E.1.2 Optimization

We train for 2,000 optimizer updates with AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$), an initial learning rate of 10^{-4}, and weight decay 10^{-4}. We use 100 linear warmup updates followed by cosine decay to 10^{-5}. Gradients are clipped to a global norm of 1.0. Backbone computation uses bfloat16, while router parameters and optimizer states are maintained in float32. Router training was performed with synchronous execution using data parallelism across eight NVIDIA RTX 5090 GPUs. For each backbone, the resulting router was reused unchanged for both synchronous and streaming inference.

E.1.3 Gating and regularization

The Gumbel-Sigmoid temperature τ is linearly annealed from 1.0 to 0.3 over the first 1,500 updates and held at 0.3 thereafter. At inference time, an edge is active when its gate probability exceeds $\xi = 0.6$. In Eq. 11, both regularizers are averaged over routing blocks and candidate edges; we set the sparsity coefficient to $\lambda_m = 10^{-3}$ and the amplitude coefficient to $\lambda_\alpha = 10^{-2}$.

E.2 Efficiency Measurement

E.2.1 Hardware and software

We measure all efficiency numbers on a single NVIDIA RTX 5090 (32 GB) with exclusive GPU access. The software stack uses PyTorch 2.9.1+cu128, pinned `transformers`, bf16 weights, and SDPA attention. Each run loads the model directly onto the benchmarked device, and no other GPU workload is run concurrently.

E.2.2 Workload

In our microbenchmark, a 1,024-token prompt is first prefilled to initialize the KV state required by each method, after which the model performs 128–192 single-token decode steps. Runtime is the end-to-end wall-clock time of the decode phase, including both backbone computation and feedback operations. We report latency as milliseconds per decoded token, averaged over the final two of three post-warm-up repetitions with explicit CUDA synchronization before and after each repetition. For Synchronous and Original Recirculation, the measurement includes advancing both the reference and feedback KV caches. Memory is measured by `torch.cuda.max_memory_allocated` after warm-up, excluding the CUDA context. Kernel counts and device time are attributed with `torch.profiler` over 8 profiled decode steps.

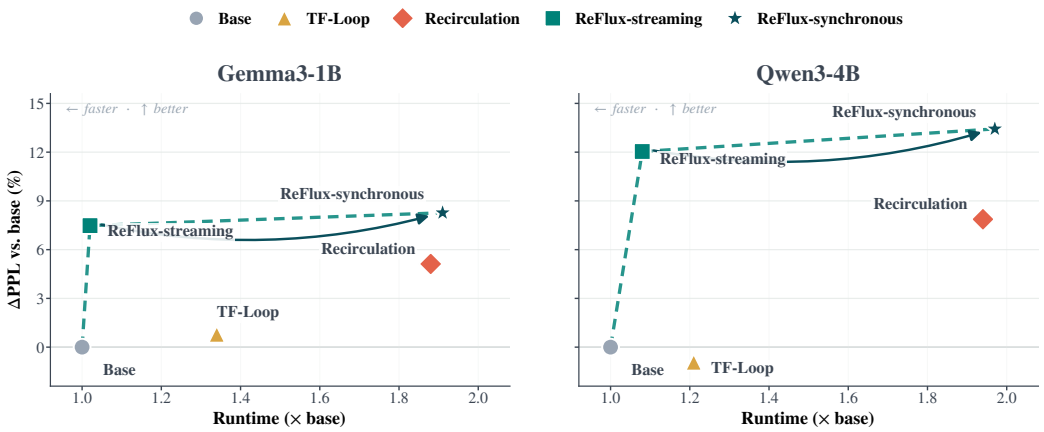


Figure 9: Pareto view of generation efficiency and quality for Gemma3-1B and Qwen3-4B. Streaming stays near the base runtime while retaining most of the PPL gain; Synchronous reaches the highest-quality frontier at roughly $2\times$ cost; Original Recirculation is dominated by the delta-based variants; and TF-Loop remains lower in quality.

E.2.3 Analytic cost and memory

We first separate the dominant backbone cost from the small auxiliary state introduced by feedback. For a generated token, the base model costs $2N$ matmul FLOPs, where N is the number of model parameters. ReFlux-streaming has the same backbone FLOPs as the base model: the feedback write is an $O(E \cdot D)$ elementwise operation and does not re-execute model weights. ReFlux-synchronous and Original Recirculation perform two full traversals and therefore cost $2\times$; more generally, an iterated variant with r feedback rounds costs $(r{+}1)\times$.

TF-Loop follows the out-of-the-box recipe recommended by its authors: a contiguous window of $W{=}4$ layers centered at fractional depth 0.5, refined by 3-stage block-mode Runge–Kutta ($K{=}3$, anchor $\beta{=}0.5$). This placement coincides with the optimum of the original window-position sweep and is shared across the two model families we evaluate. Its analytic overhead is $1 + (K{-}1)W/L$, giving $1.22\times$ at $L{=}36$ and $1.31\times$ at $L{=}26$.

Memory follows the same distinction. ReFlux changes the *contents* of KV entries but not the shape of an individual KV cache. Its persistent feedback state consists of $E \cdot D \cdot 2$ bytes of payload slots, plus transient per-step captures. During autoregressive decoding, two-traversal methods keep separate reference and feedback KV caches. Streaming advances only the feedback trajectory and therefore needs a single KV cache, so its peak memory stays essentially flat.

E.2.4 Decode-time measurements

Table 4 isolates the marginal cost of applying feedback once autoregressive decoding is underway. The remaining gap between analytic FLOPs and measured runtime is small and expected, because runtime also includes fixed per-token costs that FLOPs does not track. ReFlux-streaming adds no backbone matmuls and stays close to the base runtime ($1.02\times$ on Gemma3-1B and $1.08\times$ on Qwen3-4B). The two-traversal methods, Original Recirculation and ReFlux-synchronous, land near $2\times$ wall-clock cost. TF-Loop sits between them because it re-executes only a layer window, measuring $1.34\times$ on Gemma3-1B and $1.21\times$ on Qwen3-4B.

The small residual overhead of ReFlux-streaming comes from our hook-based prototype: the feedback write is launched from Python between existing kernel calls. With the write fused into the layer loop or captured as a CUDA graph, streaming approaches its analytic $1.00\times$ cost. By contrast, the overhead of TF-Loop and two-traversal feedback is intrinsic to re-executing model weights. Implementation improvements should therefore mainly widen the gap in favor of streaming, without changing the qualitative cost-quality ordering reported in Table 4.

Fig.9 visualizes the same tradeoff. Streaming stays closest to the base-cost corner while preserving most of the quality gain; Synchronous reaches the highest-quality frontier at roughly $2\times$ cost; Original Recirculation sits at a weaker point with similar cost; and TF-Loop, although cheaper than the two-traversal schedules, does not dominate the delta-based variants on either axis.

Table 9: Prefill/decode decomposition on a supplementary multi-hop generation workload. This table uses Gemma3-4B on 128 HotpotQA and 2WikiMultiHopQA prompts (mean length 1107 tokens), batch size 1, and 24 teacher-forced decode steps. Ratios are relative to Base within this workload.

Schedule	Prefill	Decode	Prefill (ms)	TTFT (ms)	Decode (ms/tok)	Total (s)	Prefill ×	Decode ×	E2E ×
Base	Parallel, 1 pass	Standard	69.4	104.9	32.67	0.85	1.00	1.00	1.00
Strict Stream	Sequential	Chained	35378.8	35412.1	33.03	36.17	510.08	1.01	42.38
Parallel-Prefill Stream	Parallel, 1 pass	Chained	70.1	105.9	33.76	0.87	1.01	1.03	1.02
Synchronous	Parallel, 2 pass	2 pass	134.0	201.0	66.24	1.72	1.93	2.03	2.02
Orig. Recirculation	Parallel, 2 pass	2 pass	134.7	204.8	67.36	1.75	1.94	2.06	2.05
TF-Loop	Parallel, looped window	Looped	82.5	124.1	39.26	1.02	1.19	1.20	1.20

E.2.5 Prefill and generation protocols

Autoregressive generation naturally separates into prompt prefill and token-by-token decoding. For ReFlux-streaming, this separation creates two useful operating points. *Strict streaming* applies the cached increment from position $t-1$ before computing position t, including throughout the prompt. It preserves the recurrence in Eq. 13 over the entire sequence, but necessarily serializes prompt prefill over tokens. *Parallel-prefill streaming* instead performs the prompt with the standard batched causal prefill, initializes the payload cache from the final prompt position, and activates the same chained update only during autoregressive decoding. It therefore retains token-to-token feedback during answer generation without introducing a serial prefill cost.

The two operating points answer different questions. Strict full-sequence streaming measures the quality effect of enforcing the recurrence over every scored token, including prompt tokens. Parallel-prefill streaming is the generation serving protocol: it preserves standard one-pass prompt processing and applies ReFlux on the already sequential decoding path. All ReFlux routers are trained under synchronous execution and then reused unchanged in the streaming settings, so differences between these columns reflect execution schedule rather than router retraining.

The main tables follow this distinction. Table 2 evaluates language modeling under *strict full-sequence streaming*, so its streaming PPL measures the recurrence itself rather than making a latency claim about parallel prefill. Table 3 uses *parallel-prefill streaming* for open-ended generation benchmarks (GSM8K, HotpotQA, and 2WikiMultiHopQA): the prompt is encoded once using the standard batched prefill path, and feedback is activated during autoregressive decoding. The multiple-choice benchmarks (MMLU, ARC-Easy, ARC-Challenge, HellaSwag, and Winogrande) contain no autoregressive decoding phase; candidates are selected by length-normalized conditional likelihood, so feedback can only act during the strict sequential scoring pass.

E.2.6 End-to-end prefill/decode decomposition

Table 9 complements the decode-only measurement in Table 4. Instead of isolating per-token decode overhead, it decomposes a complete prompt-to-answer request into prefill, time-to-first-token (TTFT), and decoding. This separates the cost of enforcing recurrence during prompt processing from the cost of using the same recurrence during autoregressive decoding.

The decomposition is straightforward. Strict streaming has almost no per-token decode overhead ($1.01\times$), but enforcing recurrence throughout the prompt turns prefill into a token-by-token computation. This cost is a property of the strict execution schedule, not an intrinsic decode overhead of ReFlux. Parallel-prefill streaming avoids the serialization: its prompt prefill and TTFT remain essentially at the base-model level while the same chained update is preserved during decoding. Consequently, the streaming generation results in Table 3 do not depend on serial prompt processing. Synchronous execution performs two passes and therefore incurs roughly $2\times$ prefill and decode cost; Original Recirculation has similar accounting. TF-Loop falls between these regimes at approximately $1.2\times$ end-to-end cost.

E.2.7 Recurrence Without Repeated Traversal

Taken together, these measurements show that recurrence does not have to mean another backbone traversal. Synchronous ReFlux realizes recurrence by re-executing the model, whereas ReFlux-streaming carries the feedback increment forward and incorporates it into the next token's existing traversal. Streaming therefore adds recurrent computation along the autoregressive trajectory without adding another full pass through the weights.

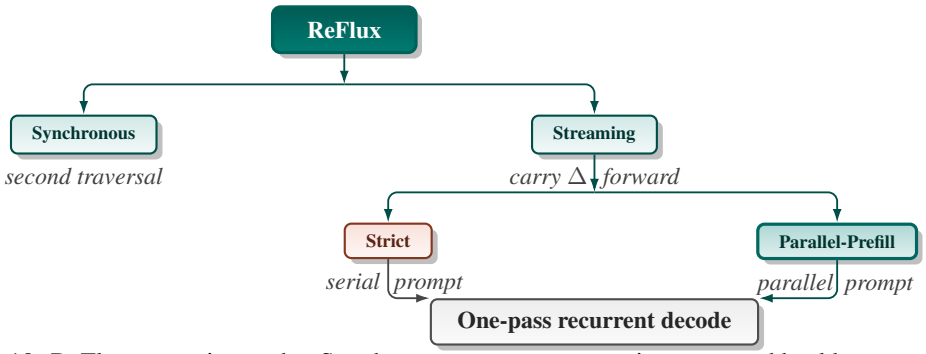


Figure 10: ReFlux execution paths. Synchronous recurrence requires a second backbone traversal, whereas streaming carries the feedback increment across tokens. The two streaming variants differ only in prompt processing and share the same one-pass recurrent decoding mechanism.

Table 10: Sensitivity to routing-block length on Gemma3-4B using ReFlux-streaming. Runtime is normalized to the base model; lower C4 PPL and higher ARC-C accuracy are better.

Block w	**C4 PPL** $\downarrow$	**ARC-C** $\uparrow$	$\bar{\kappa}$	**Runtime** ($\times$)
32	21.1	58.1	2.73	1.08
64	**21.0**	**58.2**	2.66	1.06
128	21.2	58.0	2.58	1.04
256	21.4	57.8	2.31	1.04
512	21.8	57.4	2.44	1.03

These observations make parallel-prefill streaming particularly well suited to practical generation serving. It leaves the inherently parallel prompt computation unchanged and places the recurrent update entirely on the token-by-token decoding path, where sequential computation already exists. In this sense, ReFlux does not introduce additional serialization to realize recurrence, it aligns the recurrent computation with the serialization that is already intrinsic to autoregressive decoding. The resulting execution schedule preserves standard batched prefill while augmenting each subsequent decode step with a lightweight feedback update, making parallel-prefill streaming the natural deployment operating point for ReFlux. In general, this suggests a useful systems principle for recurrent augmentation of Transformers: when recurrence can be carried along the existing autoregressive trajectory, it can provide additional computation without converting an otherwise parallel phase of inference into a sequential one.

E.3 Routing-Block Sensitivity

The block length w determines how long one routing decision is reused. We retrain the Gemma3-4B router for $w \in \{32, 64, 128, 256, 512\}$, changing no other setting. Short blocks can respond more quickly to changes in context, but call the router more often; long blocks are cheaper, but force more tokens to share the same route. Table 10 reports this trade-off under streaming execution. For a 1,024-token window, the five settings make 32, 16, 8, 4, and 2 routing decisions, respectively.

Quality changes little between $w = 32$ and 128, but begins to fall once one route is shared over 256 or more tokens. We therefore use $w = 128$ during language modeling: it stays on the quality plateau while invoking the router $4\times$ less often than $w = 32$. For benchmark evaluation, we retain $w = 128$ by default, reducing it to 32 or 4 when necessary to ensure that even the shortest inputs receive at least one context-conditioned routing step.

E.4 Routing-Threshold Sensitivity

The threshold ξ turns the router's soft gate probabilities into an active graph. We sweep $\xi \in \{0.4, 0.5, 0.6, 0.7, 0.8\}$ at inference time using the same trained Gemma3-4B router. Lower thresholds retain more candidate edges; higher thresholds make the route sparser. No retraining or other hyperparameter change is made in this comparison.

The best region is broad: moving from $\xi = 0.5$ to 0.6 removes roughly one active edge on average with almost no loss in quality. Below this range, weakly scored writes begin to interfere; above it, useful increments are discarded. We choose $\xi = 0.6$ as the sparser point in this stable region.

Table 11: Sensitivity to the inference-time routing threshold on Gemma3-4B using ReFlux-streaming. The router is not retrained between settings. Runtime is normalized to the base model.

Threshold ξ	**C4 PPL** $\downarrow$	**ARC-C** $\uparrow$	$\bar{\kappa}$	**Runtime** ($\times$)
0.4	21.5	57.6	4.61	1.06
0.5	**21.1**	**58.1**	3.37	1.06
0.6	21.2	58.0	2.58	1.04
0.7	21.5	57.7	1.74	1.03
0.8	22.0	57.1	0.91	1.03

Table 12: Routing summary by task family on Gemma3-1B. κ is the active-edge count, H is route entropy, overlap compares top-k soft-route sets, and depth span is the source-minus-target layer distance normalized by model depth.

Task family	κ	H	Same ovlp.	Cross ovlp.	Depth span
C4 / WebTextLike	2.18	1.09	0.64	0.19	0.27
BookSum / PG19	2.87	1.23	0.58	0.33	0.36
MMLU / ARC	2.33	1.14	0.67	0.30	0.24
HellaSwag / Winogrande	1.96	1.06	0.69	0.38	0.22
GSM8K	2.54	1.19	0.60	0.34	0.31
HotpotQA / 2WikiMultiHopQA	3.12	1.34	0.54	0.22	0.42
Mean	2.50	1.18	0.62	0.29	0.30

E.5 Routing Details

E.5.1 Routing statistics

We conduct all routing-behavior analyses on Gemma3-1B. Fig.6 visualizes the block-level pattern, and Table 12 summarizes the same measurements by task family. We define block difficulty as the mean token NLL ℓ_b under the unmodified backbone. Because q_{b+1} summarizes block b, we pair ℓ_b with the subsequent active-edge count κ_{b+1} and omit the first block. Panel (b) reports the activation frequency $f_e = B^{-1}\sum_{b=1}^{B} \mathbb{1}\{e \in A_b\}$ for each candidate edge. Let $\alpha_e^{(b)} = m_e^{(b)} a_e^{(b)}$ be the realized write strength of edge e in block b. At inference time, the active set is $A_b = \{e : m_e^{(b)} = 1\} = \{e : \sigma(\eta_e^{(b)}) > \xi\}$, with $\xi = 0.6$ in all reported runs. Then $\kappa_b = |A_b|$ counts active edges. To measure route concentration before hard thresholding, we define the soft write mass $w_e^{(b)} = \sigma(\eta_e^{(b)}) a_e^{(b)}$ and its normalization $\pi_e^{(b)} = w_e^{(b)} / \sum_{e' \in \mathcal{C}} w_{e'}^{(b)}$. The route entropy is $H_b = -\sum_{e \in \mathcal{C}} \pi_e^{(b)} \log \pi_e^{(b)}$. We normalize edge span by model depth and define the mean depth span as $\delta_b = \frac{1}{|A_b|}\sum_{e \in A_b}(s_e - d_e)/L$, where (s_e, d_e) are the source and target layers of edge e. For route stability, we set the overlap rank to $k = 3$ and compare the top-k soft-route sets by Jaccard overlap: $|S_b^{(k)} \cap S_{b'}^{(k)}|/|S_b^{(k)} \cup S_{b'}^{(k)}|$, where $S_b^{(k)}$ contains the k candidate edges with the largest soft write mass $w_e^{(b)}$ in block b, before hard thresholding. Same-family overlap averages this score over block pairs from the same task family, while cross-family overlap averages it over pairs from different families.

Table 12 shows that the router remains sparse but changes its support with the input. Multi-hop tasks activate the most edges and have the largest depth span, consistent with the need to propagate intermediate evidence across longer computation paths. In contrast, HellaSwag and Winogrande use fewer and shorter routes, matching their more local completion-style structure. Across all families, same-family overlap is substantially higher than cross-family overlap, indicating that related inputs reuse similar routing motifs while different task families induce different feedback patterns.

E.6 Evaluation protocol

We evaluate reasoning performance using widely adopted evaluation protocols implemented in the `lm-evaluation-harness`. We keep the prompt construction, few-shot demonstrations, and decoding settings fixed across the base and ReFlux variants, so that performance differences can be attributed to the proposed inference-time intervention. All models are evaluated as pretrained base language models without instruction tuning or chat templates. For the multiple-choice benchmarks, we use 5-shot evaluation on MMLU, 0-shot evaluation on ARC-Easy, 25-shot evaluation on ARC-Challenge, 10-shot evaluation on HellaSwag, and 5-shot evaluation on WinoGrande. For these

closed-set tasks, the prediction is the candidate with the highest length-normalized conditional log-likelihood.

For GSM8K, we follow the 5-shot generation-based evaluation protocol provided by `lm-evaluation-harness`. The model generates a solution autoregressively with greedy decoding (`temperature`=0), and the final numerical answer is extracted from the generated response for exact-match evaluation. For HotpotQA and 2WikiMultiHopQA, we use the context-grounded open-ended question-answering format described in Appendix J, without task-specific fine-tuning or external retrieval, and report exact match (EM). Unless otherwise specified, all benchmarks are evaluated on their original test or evaluation splits using the same prompt construction and decoding configuration for all model variants.

For comparisons with the original Recirculation method, we follow the evaluation protocol reported in its publicly available initial release, including the corresponding preprocessing and loss-computation procedures. The TF-Loop baseline is implemented from the recipe recommended by its authors: a contiguous window of $W{=}4$ layers centered at mid-depth is re-executed with three-stage block-mode Runge–Kutta refinement ($K{=}3$, anchor $\beta{=}0.5$). Concretely, each forward pass captures the activation entering the window, re-runs the window layers K times while collecting successive exit increments, combines them with the Runge–Kutta weights, and injects the combined update at the window exit; the layers outside the window, attention, and the key/value cache are left untouched, and no weights are updated.

F Ablations

F.1 Ablation Protocol

We conduct controlled ablations on Gemma3-4B to isolate the contributions of the feedback payload, edge-wise strength allocation, and input-conditioned routing. Unless otherwise stated, all settings use the same frozen backbone, candidate edge set, training data, optimization procedure, routing frequency, normalization operator, and inference budget as ReFlux. The ablations are trained with the same router parameterization and number of optimization steps.

Reference configuration. In the standard ReFlux configuration, each candidate edge $e = (s_e, d_e)$ carries the depth increment

$$\Delta_e(t) = z^{(0)}_{s_e}(t) - z^{(0)}_{d_e}(t),$$

where $z^{(0)}$ denotes the backbone representations before feedback. The router produces a binary gate $m^{(b)}_e$ and a non-negative amplitude $a^{(b)}_e$ for each edge at routing block b, giving

$$\alpha^{(b)}_e = m^{(b)}_e a^{(b)}_e.$$

Let $h_d(t)$ denote the current residual stream at target depth d immediately before the write. The resulting update is

$$\bar{h}_d(t) = h_d(t) + \sum_{e \in \mathcal{I}^{(b)}_d} \alpha^{(b)}_e N_{h_d(t)}(\Delta_e(t)),$$

where N denotes the same write-back normalization used by ReFlux. Thus, the payload is formed from the reference traversal, while the write is normalized against the current target stream. ReFlux jointly learns *what* to propagate through the incremental payload, *which* edges to activate, and *how strongly* to apply each active edge.

Full-state payload. **Learned-full** replaces the incremental payload with the complete source representation while leaving the routing mechanism unchanged. Specifically,

$$p_e(t) = z^{(0)}_{s_e}(t),$$

instead of

$$p_e(t) = \Delta_e(t) = z_{s_e}^{(0)}(t) - z_{d_e}^{(0)}(t).$$

The router remains input-conditioned and continues to learn both edge gates and edge-wise amplitudes. The resulting update is therefore

$$\bar{h}_d(t) = h_d(t) + \sum_{e \in \mathcal{I}_d^{(b)}} \alpha_e^{(b)} N_{h_d(t)}\left(z_{s_e}^{(0)}(t)\right).$$

This setting tests whether the benefit of ReFlux arises specifically from propagating newly accumulated computation, rather than from feeding back the entire source state.

Uniform feedback strength. **Uniform-strength** retains the incremental payload and the input-conditioned edge support, but removes edge-wise strength adaptation. Let

$$A_b = \{e : m_e^{(b)} = 1\}$$

be the active edge set at routing block b. Instead of using the learned amplitudes $a_e^{(b)}$, all active edges are assigned a common strength,

$$\alpha_e^{(b)} = \bar{\alpha}_b, \qquad e \in A_b,$$

where

$$\bar{\alpha}_b = \frac{1}{|A_b|} \sum_{e \in A_b} \alpha_e^{(b)}$$

matches the mean active-edge strength of the corresponding learned router. The support A_b is still selected by the input-conditioned gate head, and the payload remains

$$p_e(t) = \Delta_e(t).$$

Consequently, this ablation isolates the contribution of learned edge-wise strength allocation.

Fixed incremental route. **Fixed-Delta** retains both the incremental payload and learned edge-wise strengths, but removes input-conditioned changes in the edge support. We first identify a fixed edge configuration

$$E_{\text{fixed}} \subseteq C$$

from the best-performing edge configuration in the depth-geometry sweep described in Sec. 3.3. The selected edge set is then used for every input and every routing block:

$$m_e^{(b)} = \begin{cases} 1, & e \in E_{\text{fixed}}, \\ 0, & e \notin E_{\text{fixed}}. \end{cases}$$

For selected edges, the amplitude remains input-conditioned and is produced by the learned amplitude head. Hence,

$$p_e(t) = \Delta_e(t), \qquad \alpha_e^{(b)} = a_e^{(b)} \quad \text{for } e \in E_{\text{fixed}}.$$

The size of E_{fixed} is matched to the active-edge budget of ReFlux. This design isolates the value of dynamically adapting the edge support while retaining the same incremental payload and strength adaptation.

Table 13: Budget-matched adaptation baselines on Qwen3-4B. All methods use the same calibration data, optimizer, and update count; the backbone is frozen throughout. ReFlux is evaluated with synchronous execution.

	Params	Train time	C4 PPL ↓	ARC-C ↑
Base	0	–	21.6	53.8
Learned constant	1,260	1.3 h	20.9	54.2
Conditional-vector MLP	5.2M	1.8 h	19.9	54.9
LoRA-r8 (q, v)	2.9M	2.2 h	19.6	55.1
ReFlux router	2.2M	1.5 h	**18.7**	**55.8**

Geometry-based heuristic route. **Heuristic-Delta** retains the incremental payload and learned edge-wise strengths, but replaces learned input-conditioned support selection with a fixed representation-geometry heuristic. The heuristic uses only backbone representations from a calibration set and does not use router scores, downstream task labels, or evaluation data.

For each candidate edge $e = (s_e, d_e)$, we compute its average cross-depth cosine similarity on the C4 calibration set $\mathcal{D}_{\text{cal}}$:

$$C_e = \mathbb{E}_{x \sim \mathcal{D}_{\text{cal}}} \left[\cos \left(z_{s_e}^{(0)}(x), z_{d_e}^{(0)}(x) \right) \right].$$

We then define the representation deviation score

$$D_e = 1 - C_e.$$

Edges with larger D_e correspond to source–target pairs whose representations exhibit larger angular deviation across depth. We select the K candidate edges with the largest deviation scores,

$$E_{\text{heuristic}} = \text{TopK}_{e \in C} \, D_e,$$

where K is chosen to match the active-edge budget of ReFlux. The selected support is fixed across all inputs and routing blocks. For the selected edges, the payload remains

$$p_e(t) = \Delta_e(t),$$

while their amplitudes are still produced by the learned amplitude head. The heuristic is therefore a static alternative to learned route selection that directly exploits the cross-depth representation geometry identified previously.

F.2 Budget-Matched Adaptation Baselines

To separate the benefit of increment routing from the generic benefit of fitting a small module, we compare several adaptation mechanisms on Qwen3-4B. They see the same calibration windows and use the same optimizer, update count, and language-modeling objective as ReFlux; the pretrained backbone remains frozen.

1. **Learned constant mixing** learns one gate and one amplitude for each candidate edge, but uses the resulting route for every input ($2|\mathcal{C}| = 1{,}260$ parameters on Qwen3-4B).
2. **Conditional-vector mixing** follows the adaptive variant of Mozer et al. (2026), using an MLP over $[z_s; z_d]$ to produce per-dimension mixing coefficients.
3. **LoRA-r8 (Hu et al., 2021)**: rank-8 LoRA adapters on the attention q, v projections, trained with the same calibration loss.

Learned constants recover only a small part of the improvement, reducing PPL by 0.7 and raising ARC-C by 0.4 points. Conditioning the intervention or adapting attention with LoRA is stronger, but both remain behind ReFlux. In particular, ReFlux improves over LoRA by another 0.9 PPL and 0.7 accuracy points despite using fewer parameters and less training time. The comparison therefore attributes the gain not simply to adding trainable parameters, but to choosing which endogenous depth increments should be reused for the current context.

G Complete Multi-hop Case

Question and task structure

Question: *Where was the director of film Rough Going born?*
Gold answer: New Hyde Park, New York
Required chain: *Rough Going* $\xrightarrow{\text{directed by}}$ *Wally Van* $\xrightarrow{\text{born in}}$ New Hyde Park, New York.

We give the complete input and recorded outputs for the multi-hop example used in Sec. 5.4. The case is from the 2WikiMultiHopQA validation split (example 7688). Relevant passages are interleaved with distractors that share the surface words "director" and "born".

Complete context

p0 Olav Aaraas: Norwegian historian and museum director, born in Fredrikstad. *Distractor.*

p1 Peter Levin: American director of film and television. *Distractor.*

p2 Wally Van: "Wally Van (1880–1974) was an American actor and film director. He was born in New Hyde Park, New York and died in Englewood, New Jersey." *Evidence for hop 2.*

p3 Pearl Going: New Zealand socialite. *Distractor sharing "Going".*

p4 Jesse E. Hobson: director of SRI International. *Distractor.*

p5 Rough Going: "*Rough Going* is a 1925 silent film western directed by Wally Van and starring Franklyn Farnum." *Evidence for hop 1.*

p6 Joanna Going: American actress. *Distractor sharing "Going".*

p7 Ian Barry: Australian director of film and television. *Distractor.*

p8 S.N. Mathur: Director of the Indian Intelligence Bureau. *Distractor.*

p9 Brian Kennedy: art museum director and director of the Toledo Museum of Art. *Distractor.*

The two evidence passages are separated in the context: p5 identifies the director of the queried film, whereas p2 supplies that person's birthplace. The bridge entity *Wally Van* must therefore be retained while the model moves from the first passage to the second hop.

Recorded model outputs

Base model — incorrect

```
The question is asking where the director of the film Rough Going
was born.  From the context, Rough Going was directed by Wally Van.
Therefore, the relevant person is Wally Van.  Let me check again,
the director of Rough Going is indeed Wally Van.
Wally Van was an American actor and film director.
Wally Van.
```

ReFlux — correct

```
The film Rough Going was directed by Wally Van.  I therefore need
to follow the relation from Rough Going to its director, Wally Van,
and then determine where Wally Van was born.  The passage about
Wally Van states that he was born in New Hyde Park, New York.
New Hyde Park, New York.
```

The baseline reaches the bridge entity but does not complete the second hop in its recorded continuation. ReFlux produces the complete relation in its reasoning trajectory.

This behavior is consistent with prior analyses of recurrent-depth transformers, which show that intermediate entities may only become recoverable at deep layers, too late to support subsequent reasoning, and that recurrence can repair this limitation by re-executing the network (Kohli et al., 2026). ReFlux instead makes the intermediate result available at inference time by propagating the

Algorithm 1: Execution workflow of ReFlux

Input: Frozen Transformer f_θ, router g_ϕ, candidate edges $\mathcal{C}$, schedule s.
Output: Logits $\{\ell_t\}$ and payload cache $\{P_e\}_{e\in\mathcal{C}}$.

for input sequence x **do**
  /* Shared initialization */
  $P_e \leftarrow 0\ (\forall e \in \mathcal{C}), \quad \mathrm{KV} \leftarrow \emptyset$
  if $s =$ parallel-prefill stream **then**
    /* Parallel prompt prefill */
    $(\mathrm{KV}, z^{(0)}(T)) \leftarrow f_\theta(x_{\le T}; p_e = 0)$ // standard batched prefill
    $P_e \leftarrow z^{(0)}_{s_e}(T) - z^{(0)}_{d_e}(T)\ (\forall e \in \mathcal{C})$
    $q_1 \leftarrow \mathrm{Summary}(x_{\le T})$
    /* Streaming decode */
    for decode block b **do**
      $\alpha^{(b)} \leftarrow \mathrm{Route}_\phi(q_b)$ // Eqs. 8--10
      for generated token $t \in \mathcal{T}_b$ **do**
        $(\hat{z}(t), \ell_t, \mathrm{KV}) \leftarrow f_\theta(x_t; \mathrm{KV}, P, \alpha^{(b)})$ // Eq. 5
        $P_e \leftarrow \hat{z}_{s_e}(t) - \hat{z}_{d_e}(t)\ (\forall e \in \mathcal{C})$ // Eq. 13
      $q_{b+1} \leftarrow \mathrm{Summary}(\mathcal{T}_b)$
  else
    /* Full-sequence scoring */
    for routing block b **do**
      $q_b \leftarrow \begin{cases} 0, & b = 1, \\ \mathrm{Summary}(\mathcal{T}_{b-1}), & b > 1, \end{cases} \quad \alpha^{(b)} \leftarrow \mathrm{Route}_\phi(q_b)$
      if $s =$ synchronous **then**
        /* Synchronous two-pass traversal */
        $z^{(0)}(\mathcal{T}_b) \leftarrow f_\theta(x_{\mathcal{T}_b}; p_e = 0)$
        $p_e(t) \leftarrow z^{(0)}_{s_e}(t) - z^{(0)}_{d_e}(t) \quad (t \in \mathcal{T}_b,\ e \in \mathcal{C})$
        $(\hat{z}(\mathcal{T}_b), \ell_{\mathcal{T}_b}) \leftarrow f_\theta(x_{\mathcal{T}_b}; p(t), \alpha^{(b)})$ // Eq. 5
      else if $s =$ strict stream **then**
        /* Strict token-wise streaming */
        for token $t \in \mathcal{T}_b$ **do**
          $(\hat{z}(t), \ell_t, \mathrm{KV}) \leftarrow f_\theta(x_t; \mathrm{KV}, P, \alpha^{(b)})$
          $P_e \leftarrow \hat{z}_{s_e}(t) - \hat{z}_{d_e}(t)\ (\forall e \in \mathcal{C})$

newly computed increment back to an earlier, queryable state. More broadly, this mechanism is consistent with our aggregate results: ReFlux yields its strongest gains on multi-hop tasks, where later computation must access conclusions formed at earlier positions.

H ReFlux Execution Algorithm

Algorithm 1 separates the three execution paths used in the paper. In the calls above, P or $p(t)$ supplies the payloads in Eq. 5, and $\alpha^{(b)}$ supplies their block-wise write strengths. Synchronous evaluation records a no-feedback reference traversal and applies the resulting increments in a second traversal over the same block. Strict streaming applies the recurrence throughout the scored sequence. For generation, parallel-prefill streaming keeps the standard prompt prefill path and activates the same payload cache only during autoregressive decoding.

KV-cache semantics. For synchronous language-modeling evaluation, each complete window is processed twice with `use_cache=False`: the first traversal records the no-feedback reference states and the second applies their increments. Autoregressive synchronous decoding instead maintains two independent KV caches. At each step, the no-feedback traversal reads and advances only the reference cache; the feedback traversal reads and advances only the feedback cache, and its logits are used for prediction. Each trajectory therefore appends the current token exactly once, and the

reference states remain unaffected by earlier feedback writes. Streaming decoding uses only the feedback cache, since its payload at step t is taken from the feedback-augmented traversal at $t-1$.

Reference states and parallel prefill. In strict full-sequence streaming, the increment produced at position $t-1$ is injected at position t, after which the increment at t is computed from the updated stream. This keeps the backbone computation at $1.00\times$, but requires sequential processing and therefore does not exploit window-level parallelism. Parallel-prefill streaming performs one standard batched prompt traversal, retaining its KV cache and setting $P_e(T) = z^{(0)}_{s_e}(T) - z^{(0)}_{d_e}(T)$ at the final prompt position T. The first generated token consumes $P_e(T)$; every subsequent payload is then computed from the feedback-augmented decode state exactly as in Eq. 13. This schedule leaves the prompt cache unmodified and is evaluated separately from strict streaming in Table 9.

Multi-edge writes and payload cache. Active edges are applied at their target layers in increasing depth order; the additive basis $h_d(t)$ of Eq. 5 is the current stream at that depth, which already includes writes from edges targeting shallower depths. The payload cache P_e maintains slots for *all* candidate edges ($E \cdot D \cdot 2$ bytes; App. E.2) regardless of the current support, so an edge newly activated at block $b+1$ always has a previous-position payload available.

Causality. Writes at position t depend only on states at positions $\leq t$; future tokens are neither read nor written, so a perturbation of a future token leaves the logits of earlier positions unchanged by construction.

I Corpora and Datasets

I.1 Language-modeling corpora

We select ten corpora to cover a broad range of registers over which inference-time feedback may operate, from open-domain web text to highly structured and formulaic documents.

C4 (Raffel et al., 2020) is a heuristically cleaned crawl of web pages and represents the open-domain text that dominates pretraining. **LAMBADA** (Paperno et al., 2016) consists of novel passages whose final word can only be reliably predicted from the full discourse context, providing a probe of long-range coherence beyond local lexical statistics. **arXiv** and **PubMed**, two components of The Pile (Gao et al., 2020), contribute scientific and biomedical text with dense terminology, mathematical notation, and citation structure. **PG19** (Rae et al., 2019) contains long-form fiction published before 1919, while **BookSum** (Kryściński et al., 2022) provides book-length narrative text together with chapter- and book-level summaries; together, they probe coherence over substantially longer spans. **Newsroom** (Grusky et al., 2018) contains 1.3 million news articles paired with summaries from 38 major outlets. Finally, **BigPatent** (Sharma et al., 2019), comprising more than one million granted US patents, **BillSum** (Kornilova & Eidelman, 2019), which covers US legislative bills, and **GovReport** (Huang et al., 2021), which contains US government oversight and research reports, provide long and highly structured documents with sectional conventions that differ markedly from web text.

I.2 Reasoning benchmarks

The eight benchmarks cover factual knowledge, commonsense inference, multi-step reasoning, and multi-hop composition.

MMLU (Hendrycks et al., 2020) spans 57 subjects in a four-choice multiple-choice format. **ARC-Easy** and **ARC-Challenge** (Clark et al., 2018) consist of grade-school science questions, with the Challenge split designed to exclude items solvable by simple retrieval or word-co-occurrence baselines. **HellaSwag** (Zellers et al., 2019) contains adversarially filtered sentence-completion problems drawn from physical and instructional contexts, while **Winogrande** (Sakaguchi et al., 2021) extends Winograd-style pronoun resolution through contrastive twin pairs. **GSM8K** (Cobbe et al., 2021) consists of grade-school mathematical word problems requiring multi-step reasoning.

The final two benchmarks, **HotpotQA** (Yang et al., 2018) and **2WikiMultiHopQA** (Ho et al., 2020), directly probe the composition behavior targeted by our method. Both require multi-hop question answering over Wikipedia-style passages, where relevant evidence is distributed across documents and intermediate conclusions must remain available for subsequent reasoning steps. This setting

provides a mechanistically informative test of ReFlux, as preserving intermediate information across depth is precisely the behavior that our feedback mechanism is designed to support.

J PROMPT TEMPLATES

All models are evaluated as pretrained base language models without instruction tuning or chat templates. We use the prompt formats of the corresponding benchmark implementations, following the widely adopted `lm-evaluation-harness` conventions. The same prompt construction and decoding protocol are used for the base and ReFlux variants. For few-shot evaluation, demonstrations are placed before the target example using the benchmark's standard formatting.

MMLU. We use the subject-specific multiple-choice format adopted by the benchmark implementation. Each demonstration contains the question, four answer choices, and the ground-truth answer, while the target example omits the answer.

MMLU

```
The following are multiple choice questions (with answers) about
subject.
question
A. choice A  B. choice B  C. choice C  D. choice D
Answer:  answer

target question
A. choice A  B. choice B  C. choice C  D. choice D
Answer:
```

We use 5-shot evaluation. Predictions are obtained by comparing the conditional likelihoods of the candidate answer labels, and we report accuracy.

ARC-Easy. Following the `lm-evaluation-harness` task definition, the question and answer prefix form the input prompt, while the candidate choices are scored separately by the multiple-choice evaluation interface.

ARC-Easy

```
Question:  question Answer:
```

We use 0-shot evaluation and select the candidate answer with the highest conditional likelihood. We report accuracy.

ARC-Challenge. ARC-Challenge uses the same prompt construction as ARC-Easy, with a different number of few-shot demonstrations.

ARC-Challenge

```
Question:  question Answer:
```

We use 25-shot evaluation and select the candidate answer with the highest conditional likelihood. We report accuracy.

HellaSwag. For HellaSwag, the processed context constitutes the prompt, while the candidate endings are evaluated as alternative continuations.

HellaSwag

```
context / query
```

For each example, the model assigns a conditional likelihood to each candidate ending, and the candidate with the highest score is selected. We use 10-shot evaluation and report accuracy.

WinoGrande. WinoGrande is evaluated as a two-choice sentence-completion task. The prompt contains the sentence with its blank, while the two candidate fillers are scored as alternative continuations.

WinoGrande

```
sentence with blank
```

We use 5-shot evaluation and select the candidate with the highest conditional likelihood, and report accuracy.

GSM8K. We follow the 5-shot generation-based evaluation protocol provided by `lm-evaluation-harness`. Each demonstration contains a question and its reference solution, while the target example ends with an empty answer field.

GSM8K

```
Question: demonstration question Answer: demonstration solution and
final answer

Question: target question Answer:
```

The model generates a solution autoregressively with greedy decoding (`temperature`=0), and the final numerical answer is extracted from the generated response for exact-match evaluation. We use 5-shot evaluation.

HotpotQA. For HotpotQA, we use a context-grounded multi-hop question-answering format with the benchmark-provided context. The context is presented before the question, and the model is asked to produce the final answer directly.

HotpotQA

```
Answer the question based on the given paragraphs.
The following are given paragraphs.
context
Answer the question based on the given paragraphs.
Question: question Answer:
```

We use 0-shot evaluation without task-specific fine-tuning or external retrieval, and report exact match (EM).

2WikiMultiHopQA. We use the same context-grounded question-answering format for 2WikiMultiHopQA.

2WikiMultiHopQA

```
Answer the question based on the given paragraphs.
The following are given paragraphs.
context
Answer the question based on the given paragraphs.
Question: question Answer:
```

We use 0-shot evaluation without task-specific fine-tuning or external retrieval, and report exact match (EM).