LATENT RECIRCULATION: REFINING REASONING THROUGH GATED LATENT FEEDBACK

Anonymous authors
Paper under double-blind review

ABSTRACT

Chain-of-thought (CoT) reasoning improves the performance of language models by making intermediate steps explicit, but decoding these steps autoregressively incurs substantial output overhead. Latent reasoning instead performs intermediate computation in continuous representations, offering a compact alternative to explicit thought generation. However, constructing useful latent updates and supervising their evolution remain challenging. We introduce *Latent Recirculation*, a lightweight framework that constructs a fixed, question-conditioned latent workspace and refines it through gated deep-to-shallow feedback. The feedback allows the reasoner to revisit the same workspace without increasing its length or generating intermediate text. To guide this refinement, an auxiliary explicit CoT branch provides a reference state during training, and a transition objective aligns latent updates with the displacement toward that reference. Both pretrained backbones remain frozen, with optimization restricted to the latent interface and low-rank adapters on the reasoner. Experiments on symbolic, arithmetic, and commonsense reasoning benchmarks show improved average accuracy across two reasoners, while ablations highlight the importance of combining feedback with transition supervision.

1 INTRODUCTION

Large language models can improve reasoning performance by generating intermediate steps. Chain-of-thought (CoT) prompting makes this process explicit (Wei et al., 2022; Kojima et al., 2022), but decoding a long reasoning trace token by token incurs substantial latency and output overhead. Latent reasoning instead performs intermediate computation in continuous representations. Coconut (Hao et al., 2024) generates continuous thoughts autoregressively, CODI (Shen et al., 2025) distills explicit reasoning into a continuous path, and SoftCoT (Xu et al., 2025) and SemCoT (He et al., 2025) use lightweight latent construction and semantic guidance to make these representations informative. These methods provide compact representations, but their effective refinement remains an open problem.

Looped computation increases effective depth by reusing parameters (Dehghani et al., 2018), and later work connects looping to iterative algorithms and latent reasoning (Yang et al., 2024; Giannou et al., 2023; Xu & Sato, 2024; Saunshi et al., 2025). PCCoT (Wu et al., 2025) and LOTUS (Fan et al., 2026) apply iterative computation to latent representations, and Recirculation (Mozer et al., 2026) feeds contextualized deep representations back into shallower computation while preserving pretrained weights, whose effectiveness with a single additional iteration motivates our feedback principle.

Constructing useful updates for a compact workspace remains difficult. Autoregressive latent generation adds a sequential dependency between positions, whereas enlarging the workspace only increases its length and repeating an unchanged deterministic computation does not refine its states. This leads to **Challenge 1**: how to update a compact latent workspace as a whole, refining its positions together without increasing the workspace length or generating intermediate thoughts.

Supervising the evolution of these latent states is equally challenging. Answer supervision constrains only the final prediction, not the direction of individual updates, while tying each position

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.

to a specific CoT token restricts how reasoning is distributed across the workspace. This leads to **Challenge 2**: how to guide the refined latent states toward useful representations without a token-by-token correspondence or intermediate reasoning text at inference.

In this paper, we propose *Latent Recirculation*, a lightweight framework that refines a compact latent workspace through a single gated feedback pass. For **Challenge 1**, Thought Projection initializes the workspace from a frozen student, and *Gated KV Recirculation* uses the reasoner's contextualized latent states to refine all positions together through deep-to-shallow attention feedback, adding computation at a fixed latent length. For **Challenge 2**, *CoT-Guided Transition Alignment* supervises each update by the displacement toward a reference state from an auxiliary explicit CoT branch, without assigning a reasoning token to any latent position. The CoT branch is removed at inference, and training is restricted to the latent interface and reasoner adapters while both pretrained backbones remain frozen. Our contribution is a latent-specific recirculation formulation that couples fixed-workspace feedback with explicit transition-level supervision.

Our contributions are summarized as follows:

- We present the Latent Recirculation framework, which constructs a fixed, question-conditioned latent workspace and refines it through one additional feedback pass.
- We propose Gated KV Recirculation to jointly refine latent positions through an additional feedback pass, without extending the workspace or generating intermediate text.
- We propose CoT-Guided Transition Alignment to guide latent updates through a CoT-conditioned displacement without token-level reconstruction.
- We evaluate the framework on symbolic, arithmetic, and commonsense reasoning benchmarks, and analyze its components, feedback depth, and sensitivity to refinement settings.

2 Related Work

2.1 Latent and Implicit Reasoning

Implicit reasoning reduces or removes intermediate textual generation. Pause tokens provide additional computation before answering (Goyal et al., 2024), while stepwise internalization gradually removes explicit reasoning steps during training (Deng et al., 2024). Coconut (Hao et al., 2024) and CODI (Shen et al., 2025) instead generate continuous thoughts sequentially, with CODI transferring information from an explicit reasoning branch through self-distillation.

Other methods focus on constructing and supervising continuous representations. SoftCoT (Xu et al., 2025) uses a lightweight assistant and a projection to provide thought features, and SemCoT (He et al., 2025) adds semantic guidance to implicit generation. CoLaR (Tan et al., 2025) compresses reasoning tokens into latent representations and refines them with reinforcement learning. PCCoT (Wu et al., 2025) updates a fixed latent block using parallel Jacobi iterations.

Supervising the latent states is itself an active question. Some methods tie the latent states to the reasoning trace (Hao et al., 2024; Fan et al., 2026), whereas others match the implicit reasoning to the explicit trace at the level of meaning (He et al., 2025), distil a compact generator from a stronger model (Xu et al., 2025), or refine compressed representations with reinforcement learning (Tan et al., 2025). A related line observes that models can revise earlier answers, which motivates supervision that acts on how a state changes rather than only on the final prediction (Huang et al., 2024).

These methods also differ in cost. Explicit CoT pays in output tokens, since the trace is usually longer than the answer. Latent methods reduce that cost, but not uniformly: autoregressive latent chains remain sequential, methods that build the workspace with a separate generator pay for that generator, and looped designs may pass over the backbone several times per example. Output length alone therefore does not determine end-to-end latency, which also depends on the number of passes over the backbone and on how each pass uses the cache.

2.2 Looped Transformers and Recirculation

Universal Transformers (Dehghani et al., 2018) reuse a shared transition across depth. Subsequent work studies iterative algorithms and the expressive power of looped computation (Yang et al., 2024;

Giannou et al., 2023; Xu & Sato, 2024; Saunshi et al., 2025). Relaxed Recursive Transformers (Bae et al., 2025) convert pretrained models through parameter sharing and layer-wise adapters, while recurrent-depth language models learn latent computation during pretraining (Geiping et al., 2025). LOTUS (Fan et al., 2026) applies looped computation to latent reasoning and supervises looped positions with corresponding CoT tokens.

Recirculation (Mozer et al., 2026) mixes deep representations into shallower layers across both depth and input steps. Its reported experiments use one additional iteration with the original weights frozen. We adopt this mixing principle with a different schedule: a fixed prefix is processed again, and only first-layer latent keys and values receive feedback. This adds a prefill pass, unlike the original token-by-token recurrence. Our directional objective supervises state updates through a CoT-conditioned displacement.

These approaches differ from ours in the object and the supervision of recurrence. PCCoT iteratively updates latent variables through Jacobi-style parallel computation, and LOTUS associates looped latent positions with explicit CoT tokens, while Recirculation introduces deep-to-shallow feedback but does not formulate it as a supervised revision of latent states. In contrast, our feedback operates on a fixed reasoning workspace and is explicitly trained at the level of state transitions, without token-level correspondence. Our method distinguishes itself by coupling deep-to-shallow feedback over a fixed latent workspace with transition-level supervision, without requiring token-level correspondence.

A parallel line analyzes what looped computation buys. Looped and recurrent-depth models can emulate iterative algorithms and, with enough iterations, approximate programs whose depth grows with the number of steps (Giannou et al., 2023; Yang et al., 2024), and analyses of latent reasoning show that looping raises the effective depth available to a fixed parameter budget (Saunshi et al., 2025; Xu & Sato, 2024). These results are stated for computation that is repeated many times, so they describe an upper bound rather than what a particular training procedure reaches. Our setting is the complementary regime: a pretrained model is reused without changing its weights, and the question is how much one additional pass adds. Unlike recurrent-depth approaches that study scaling computation through repeated iterations, we focus on whether a lightweight feedback pass can effectively refine a fixed latent workspace in a pretrained reasoner.

3 METHOD

3.1 OVERVIEW

Given a question q, the model predicts an answer a; a reference CoT trace c is available only during training. Our method carries out the reasoning computation in a compact latent workspace and refines it with one additional feedback pass. As shown in Figure 1, it consists of three components: Thought Projection initializes the workspace, Gated KV Recirculation refines it, and CoT-Guided Transition Alignment supervises how it changes. The last component is used only during training.

Throughout, $[a; b]$ denotes concatenation, and we distinguish two objects that share the same m positions: the *input workspace* Z, computed once and kept fixed, and the *latent state* S^r, containing the reasoner's final-layer hidden states at those positions after pass r. We use $R = 1$ by design; $R > 1$ is used only for diagnostic analysis.

3.2 THOUGHT PROJECTION

We build on two pretrained language models: a small feature extractor g_ϕ (the student) that initializes latent positions, and a larger reasoner $f_{\theta,\eta}$ (the teacher) that processes them and decodes the answer. Here ϕ and θ are frozen backbone weights, while η denotes trainable low-rank adapters (LoRA) (Hu et al., 2021) on the reasoner's query and value projections. LoRA is used only on the reasoner: the student is taken as-is, and no adapters are trained on it. Besides η, training updates only the projection ψ and a new thought embedding e_{th}.

The student reads the question followed by m `<thought>` slots in one forward pass. These slots are new to the student, whose embedding table is frozen, so their input embedding is the trainable vector $e_{\text{th}} \in \mathbb{R}^{d_s}$. Let $\mathcal{I}_s$ index these positions and d_s be the student's hidden size. The projected

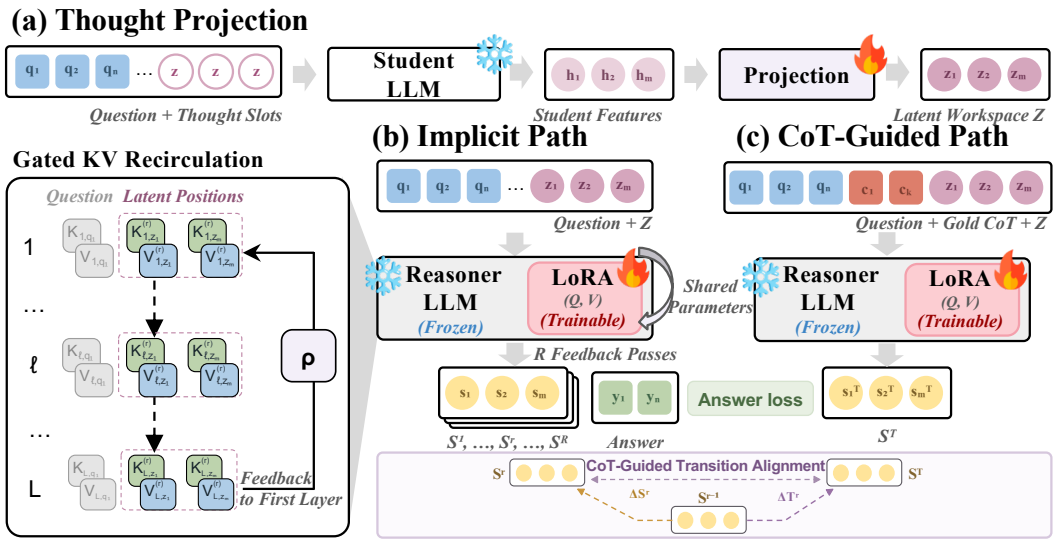


Figure 1: Latent Recirculation. (a) Thought Projection builds the fixed workspace Z from frozen student features. (b) Gated KV Recirculation refines its latent states via one deep-to-shallow feedback pass ($R = 1$), and the final cache decodes the answer. (c) A training-only CoT-guided path supplies the reference state S^T for CoT-Guided Transition Alignment. Both paths share the reasoner and its trainable adapters, while both pretrained backbones stay frozen.

block supplies the embeddings of m contiguous reasoner positions after the question:

$$H_q = g_\phi([q; e_{\text{th}}^m])_{\mathcal{I}_s} \in \mathbb{R}^{m \times d_s}, \qquad Z = P_\psi(H_q) \in \mathbb{R}^{m \times d}, \tag{1}$$

where P_ψ projects the student features to the reasoner's hidden size d. The student observes neither c nor a. Producing all latent positions in one pass avoids autoregressive latent generation and leaves the student's causal attention unchanged. The block Z is the same in every pass: it is never overwritten or extended.

3.3 Gated KV Recirculation

A first reasoner pass over $[q; Z]$ yields the initial latent state

$$S^0 = f_{\theta,\eta}([q; Z])_{\mathcal{I}} \in \mathbb{R}^{m \times d}, \tag{2}$$

where $\mathcal{I}$ indexes the m latent positions. We then perform R further passes. In each of them, the latent state of the preceding pass is fed back into the first decoder layer (layer 0).

Let h_i^r be the input to layer 0's key and value projections at latent position i in pass r. In a pre-normalized decoder, h_i^r is the normalized layer input, rather than the raw embedding. Since the input block is fixed, this local input is identical across deterministic passes. We use its norm as the reference scale and project the rescaled state with the reasoner's frozen base weights:

$$\widetilde{S}_i^{r-1} = \frac{\|h_i^r\|_2}{\max(\|S_i^{r-1}\|_2, \epsilon_{\text{mem}})} S_i^{r-1}, \qquad \widehat{K}_i^r = W_K \widetilde{S}_i^{r-1} + b_K, \qquad \widehat{V}_i^r = W_V \widetilde{S}_i^{r-1} + b_V. \tag{3}$$

Here W_K, b_K and W_V, b_V are the reasoner's pretrained key and value parameters, so the feedback path uses the frozen base weights, whereas the local value projection also includes its LoRA update; the key projection has no adapter. For positions with $\|S_i^{r-1}\|_2 \le \epsilon_{\text{mem}}$, we bypass the mixture and retain the local keys and values. The local projections $K_{\text{loc},i}^r$ and $V_{\text{loc},i}^r$, with the value projection including its LoRA update, are then blended with the projected state through a single scalar ρ:

$$K_i^r = (1-\rho) K_{\text{loc},i}^r + \rho \widehat{K}_i^r, \qquad V_i^r = (1-\rho) V_{\text{loc},i}^r + \rho \widehat{V}_i^r. \tag{4}$$

The same fixed $\rho \in [0, 1]$ is shared across positions, key and value streams, and passes, and we select it on validation data. Only the keys and values of latent positions at layer 0 are modified: the feedback hook does not modify queries or the projections at question and answer positions, and the

remaining layers use the standard adapted computation to produce S^r. Keys receive their ordinary positional encoding after mixing.

Because Z and the sequence length are fixed, the feedback adds computation depth rather than context length, and all latent positions are updated within the same pass. During training, the last feedback pass is the teacher-forced answer pass described below. At inference, we run the initial pass and R feedback passes as a prefill and decode the answer from the resulting cache, so all R feedback passes occur during prefill, with no additional feedback pass during answer decoding.

3.4 CoT-Guided Transition Alignment

The implicit path never observes the trace. To supervise how the state changes, a training-only branch processes $[q; c; Z]$ once with feedback disabled, so that its latent positions attend to the complete trace under the causal mask. The two blocks are assigned positional indices such that the latent positions follow the trace, so their order in the notation does not affect the computation. We take the final-layer states of those positions as a target and stop the gradient through this branch,

$$S^T = \mathrm{sg}(f_{\theta,\eta}([q; c; Z])_{\mathcal{I}_c}), \tag{5}$$

where sg denotes stop-gradient and $\mathcal{I}_c$ selects the m latent positions that follow the trace. The target is recomputed with the current adapters η but contributes no gradient; positions in the two branches are matched by index, and no separate target network is trained.

We want each pass to move the state in the direction that the trace would induce, without requiring the state to match the target exactly. With the realized update and the target direction for latent position i at pass r defined as

$$\Delta S_i^r = S_i^r - S_i^{r-1}, \qquad \Delta T_i^r = S_i^T - \mathrm{sg}(S_i^{r-1}), \tag{6}$$

we minimize the angular difference between them, averaged over the R passes and the m latent positions,

$$\mathcal{L}_{\mathrm{tr}} = \frac{1}{R\,m} \sum_{r=1}^{R} \sum_{i=1}^{m} \left(1 - \frac{\langle \Delta S_i^r, \Delta T_i^r \rangle}{\max(\|\Delta S_i^r\|_2, \epsilon_{\cos}) \max(\|\Delta T_i^r\|_2, \epsilon_{\cos})} \right). \tag{7}$$

We define the loss to be zero when $R = 0$. The objective constrains the direction rather than the magnitude of each update, allowing the answer objective to determine the appropriate update scale while encouraging the transition to follow the CoT-conditioned displacement. Gradients flow through ΔS_i^r, while the reference direction is detached.

3.5 Training and Inference

The answer is predicted on the implicit path only. From the cache $\mathcal{C}^R$ produced by the final pass, we take the answer loss $\mathcal{L}_{\mathrm{ans}}$ to be the token-level cross-entropy over the answer tokens, and the total objective is

$$\mathcal{L} = \alpha \mathcal{L}_{\mathrm{tr}} + (1 - \alpha) \mathcal{L}_{\mathrm{ans}}, \tag{8}$$

with $\alpha \in [0, 1)$ balancing the two terms. Only answer tokens contribute to $\mathcal{L}_{\mathrm{ans}}$; question, thought, and answer-cue positions are masked. When $R = 0$, we optimize $\mathcal{L}_{\mathrm{ans}}$ alone. Optimization updates $\{\psi, e_{\mathrm{th}}, \eta\}$ and leaves both backbones frozen.

Under teacher forcing, the final pass processes the prefix and the answer suffix together and therefore yields both S^R and the answer logits, while causal attention keeps answer tokens out of the latent states. The CoT branch adds one training-only forward. At inference the trace and the target branch are dropped, and the model decodes the answer from $\mathcal{C}^R$ without generating intermediate text.

4 Experiments

4.1 Evaluation Setup

Tasks and metrics. We evaluate on five benchmarks, following the suite and splits of the SemCoT release (He et al., 2025). CoinFlip measures symbolic state tracking; GSM8K (Cobbe et al., 2021),

MultiArith (Roy & Roth, 2015), and SVAMP (Patel et al., 2021) measure arithmetic reasoning; and CommonsenseQA (Talmor et al., 2019) (COMMON) measures commonsense reasoning. Every method uses the same training and evaluation subsets fixed by the release, and all methods are scored with the same final-answer exact-match protocol, detailed in Appendix B.

Models and baselines. We use Llama-2-7B (Touvron et al., 2023) and Mistral-7B (Jiang et al., 2023) as reasoners, paired with frozen student feature extractors. We compare against explicit generation (CoT), a pause-token baseline (Pause (Goyal et al., 2024)), latent-reasoning methods (ICoT-SI (Deng et al., 2024), Coconut (Hao et al., 2024), CODI (Shen et al., 2025), SoftCoT (Xu et al., 2025), SemCoT (He et al., 2025), CoLaR (Tan et al., 2025), and PCCoT (Wu et al., 2025)), and a looped reference (LOTUS (Fan et al., 2026)). Comparisons share the reasoner checkpoint and tokenizer, training and evaluation splits, checkpoint-selection criterion, and final-answer evaluation protocol. Each baseline retains its method-specific objective and hyperparameter configuration, as detailed in Appendix B. Because the methods differ in their trainable modules and in the computation they add, we also report trainable parameters, training computation, and inference latency in Table 3; Appendix B lists what is shared and what is method-specific.

Configuration and uncertainty. The main configuration uses $m = 12$, $R = 1$, and $\alpha = 0.25$, with ρ selected on validation data. For trained methods, we report mean accuracy and standard deviation over five independently trained runs on shared evaluation examples. For zero-shot CoT, which involves no training, we report it only as an explicit-reasoning reference under the same final-answer evaluation protocol.

4.2 MAIN RESULTS

Table 1 compares explicit CoT generation with pause-token and latent-reasoning methods. Latent Recirculation achieves the highest five-task average on both backbones, with particularly clear improvements on CoinFlip and CommonsenseQA for Llama-2-7B and on SVAMP for Mistral-7B. Explicit CoT serves as a protocol-matched reference for explicit reasoning generation and is prompted zero-shot, whereas the latent methods are trained on the task; Appendix B further details this distinction and their inference costs. Since all methods are evaluated under the same final-answer protocol, we focus on comparisons within this controlled setting. Overall, the results support supervised refinement of a compact latent workspace as a useful alternative to intermediate textual decoding.

4.3 COMPONENT ABLATIONS

Table 2 reports retrained variants that differ in key components under the same data and selection budget. No feedback removes both recirculation and its associated transition objective, and therefore measures their combined contribution. No alignment retains the feedback pass but removes transition supervision, isolating whether recirculation alone is sufficient. Direct CoT alignment retains feedback but replaces the directional objective with direct matching to the CoT-conditioned reference state, thereby supervising where the latent state should be rather than how it should move. The remaining variants remove the reasoner adapters or the norm matching of the recirculated state.

The full method achieves the highest average accuracy on both backbones. Retaining feedback while removing its transition supervision lowers the five-task average below the no-feedback variant on both backbones (36.67 vs. 37.69 on Llama-2-7B; 39.72 vs. 44.13 on Mistral-7B), suggesting that the aggregate benefit of recirculation depends strongly on how its state updates are supervised. Direct state matching partially recovers this loss but remains consistently below directional transition alignment, supporting supervision of how the latent state evolves rather than only where it ends. The effects of reasoner adaptation and norm matching are backbone-dependent and account for the largest single drops on each backbone: removing reasoner LoRA is most harmful on Llama-2-7B (-4.47), while removing norm matching is most harmful on Mistral-7B (-7.35). Overall, the results indicate that the feedback interface and transition supervision contribute jointly to the full model.

4.4 FEEDBACK DEPTH

We ask whether naively repeating the same shared feedback rule provides additional benefit. We train a separate Llama-2-7B model at each depth $R \in \{0, 1, 2\}$ and evaluate it at the same depth, so

Table 1: Reasoning accuracy (%; mean ± standard deviation over five runs). AVG. is the unweighted mean over CoinFlip, GSM8K, MultiArith, SVAMP, and CommonsenseQA (COMMON). Bold and underline mark the best and second-best means among non-explicit methods; italicized explicit CoT is excluded from that ranking. All methods are evaluated using the same final-answer exact-match protocol; results should therefore be interpreted within this controlled evaluation setting rather than compared directly with numbers reported under different evaluation protocols.

Method	CoinFlip	GSM8K	MultiArith	SVAMP	Common	Avg.
Llama-2-7B reasoner						
Explicit CoT	47.90 ± 0.80	14.60 ± 1.39	43.33 ± 1.22	38.50 ± 0.89	50.40 ± 1.74	38.95
Pause	53.10 ± 0.55	$\mathbf{6.80 \pm 1.30}$	$\underline{14.56 \pm 3.86}$	$\underline{44.30 \pm 1.10}$	60.22 ± 0.30	$\underline{35.80}$
ICoT-SI	49.90 ± 4.77	3.00 ± 1.00	4.44 ± 1.36	15.90 ± 1.56	48.90 ± 1.98	24.43
Coconut	50.00 ± 2.30	2.10 ± 0.49	2.89 ± 1.55	43.00 ± 1.38	51.90 ± 1.59	29.98
CODI	50.70 ± 2.02	1.30 ± 1.15	3.00 ± 0.84	3.80 ± 1.69	19.80 ± 2.68	15.72
SoftCoT	50.20 ± 2.36	3.00 ± 0.87	5.44 ± 2.65	12.30 ± 2.51	54.80 ± 0.76	25.15
SemCoT	$\underline{55.50 \pm 2.15}$	3.20 ± 0.91	3.67 ± 0.93	19.60 ± 2.90	51.90 ± 1.47	26.77
CoLaR	52.50 ± 3.62	4.50 ± 0.55	8.33 ± 1.36	42.90 ± 0.42	$\underline{62.30 \pm 2.23}$	34.11
PCCoT	52.00 ± 2.00	5.89 ± 0.90	12.00 ± 2.77	43.80 ± 1.25	60.90 ± 2.22	34.92
LOTUS	48.50 ± 0.00	2.00 ± 0.61	3.33 ± 0.56	24.00 ± 2.06	54.80 ± 2.86	26.53
Ours	$\mathbf{62.20 \pm 0.68}$	$\underline{6.30 \pm 1.44}$	$\mathbf{17.56 \pm 2.78}$	$\mathbf{46.20 \pm 0.81}$	$\mathbf{69.30 \pm 0.87}$	**40.31**
Mistral-7B reasoner						
Explicit CoT	39.60 ± 2.96	10.50 ± 0.87	44.89 ± 1.13	36.90 ± 0.86	61.90 ± 0.22	38.76
Pause	49.30 ± 0.75	8.50 ± 1.17	14.56 ± 3.45	$\underline{54.20 \pm 1.04}$	56.00 ± 1.55	36.51
ICoT-SI	45.90 ± 2.52	8.60 ± 1.91	6.67 ± 0.22	14.90 ± 1.93	60.60 ± 1.14	27.33
Coconut	48.80 ± 1.69	0.80 ± 0.40	1.78 ± 0.42	41.90 ± 0.80	72.70 ± 0.40	33.20
CODI	50.00 ± 1.90	10.20 ± 2.51	24.78 ± 1.08	53.80 ± 2.17	$\mathbf{78.10 \pm 0.55}$	43.38
SoftCoT	42.80 ± 0.57	10.30 ± 1.60	8.78 ± 2.27	28.80 ± 0.27	74.90 ± 0.89	33.12
SemCoT	46.00 ± 3.14	7.00 ± 1.87	7.67 ± 0.72	18.50 ± 1.41	63.20 ± 0.57	28.47
CoLaR	50.10 ± 1.08	10.20 ± 2.11	6.70 ± 1.04	48.90 ± 1.32	76.10 ± 0.86	38.40
PCCoT	$\underline{52.00 \pm 2.00}$	$\underline{10.40 \pm 2.70}$	$\mathbf{33.11 \pm 0.84}$	48.00 ± 1.54	75.10 ± 2.07	$\underline{43.72}$
LOTUS	45.10 ± 1.78	6.00 ± 0.79	7.33 ± 0.91	45.20 ± 1.96	75.80 ± 1.48	35.89
Ours	$\mathbf{52.10 \pm 2.33}$	$\mathbf{11.30 \pm 1.47}$	$\underline{29.67 \pm 2.55}$	$\mathbf{55.70 \pm 2.36}$	$\underline{77.50 \pm 1.64}$	**45.25**

Table 2: Component ablations. Entries are five-task mean accuracy and the change from the full method in percentage points. Each backbone has its own full-method reference, and every variant is retrained with the same data and selection budget.

	Llama-2-7B		Mistral-7B	
Configuration	Avg.	Δ	Avg.	Δ
Full method	40.31	0.00	45.25	0.00
No feedback	37.69	−2.62	44.13	−1.12
No alignment	36.67	−3.64	39.72	−5.53
Direct CoT alignment	37.82	−2.49	42.33	−2.92
No reasoner LoRA	35.84	−4.47	43.90	−1.35
No norm matching	38.48	−1.83	37.90	−7.35

the comparison is about the depth chosen for training, not about scaling a fixed model at inference (Figure 2).

A single feedback pass performs best under the current shared-update parameterization. While $R = 1$ improves the five-task average over $R = 0$, further increasing the depth to $R = 2$ does not yield additional gains, suggesting that effective multi-pass refinement may benefit from pass-specific update mechanisms. Single-pass recirculation is the intended operating regime; deeper recurrence would require a different, depth-aware design.

4.5 FEEDBACK STRENGTH, SUPERVISION, AND CAPACITY

We study three hyperparameters, varying one at a time while holding the others fixed and training each setting independently: the feedback strength ρ, which sets how much of the recirculated state is mixed in; the transition-loss weight α; and the latent length m (Figure 3).

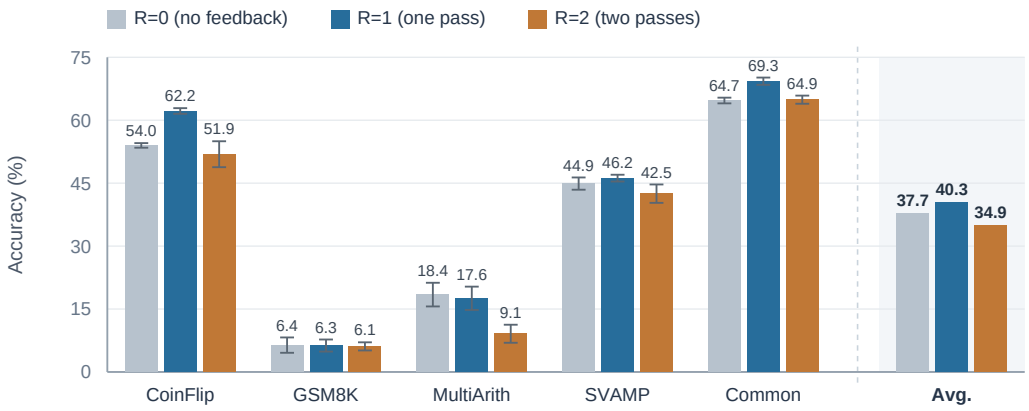


Figure 2: Llama-2-7B accuracy by feedback depth. Models are trained and evaluated at the same depth. Bars show task accuracies and their unweighted average; error bars denote standard deviations over five runs.

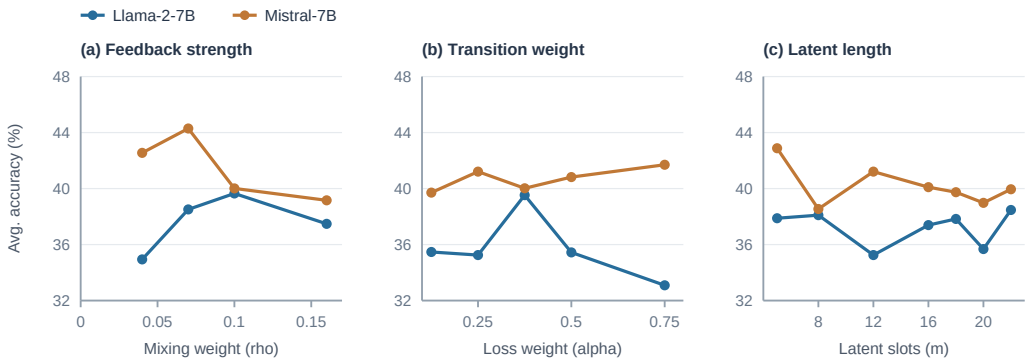


Figure 3: Sensitivity of five-task average accuracy to (a) feedback strength ρ, (b) transition-loss weight α, and (c) latent length m. Each setting is trained independently.

Feedback strength has a clear optimum. Accuracy is highest at an intermediate ρ on both backbones and falls when the mixing is stronger, so useful refinement needs the recirculated state to inform the update without overwriting the local one. The best value is not the same for the two backbones, so ρ is selected on validation data. The transition weight α matters more on Llama-2-7B, where accuracy varies across the sweep, than on Mistral-7B, which stays roughly flat. Latent length m shows no strong dependence: performance remains relatively stable across latent lengths, indicating that the method does not rely on enlarging the latent workspace, and Mistral-7B performs best with a compact workspace. Across all three, accuracy depends on how the feedback is configured rather than on simply making it stronger or larger.

This sweep fixes a single ρ across tasks, so its absolute accuracies are not directly comparable to those of the main configuration.

4.6 Training Cost and Efficiency

We report cost from two angles: what a method costs to adapt, and what it costs to run. Table 3 covers adaptation and output length, listing the trainable parameters, the training computation, and the mean generated answer length; it carries no accuracy column, since the accuracy of the same methods is in Table 1 and the two are read together. Figure 4 covers inference, plotting the five-task average against the median GSM8K latency.

Our method is cheap to adapt. Training updates only the projection, the thought embedding, and the reasoner adapters, so it has the fewest trainable parameters and the lowest training computa-

Table 3: Training cost and adaptation footprint of the selected configuration (cost only; accuracy of the same methods is in Table 1). Trainable parameters exclude the frozen backbones; training computation counts forward and backward passes up to the selected checkpoint, including auxiliary branches (M: 10^6 parameters; P: 10^{15} FLOPs). Answer tokens are the mean generated length on GSM8K at batch size one; the latency of the same runs is shown in Figure 4. Bold and underline mark the best and second-best entry in each row, where lower is better.

	Coconut	CODI	SemCoT	CoLaR	PCCoT	LOTUS	**Ours**
Llama-2-7B							
Trainable (M)	270.56	304.12	1288.28	<u>41.96</u>	270.56	270.56	**12.59**
Train FLOPs (P)	165.87	54.17	102.15	<u>50.75</u>	99.10	83.88	**48.47**
Answer tok.	6.77	6.0	30.0	**4.31**	4.86	<u>4.32</u>	**4.31**
Mistral-7B							
Trainable (M)	268.98	302.55	1093.71	<u>40.38</u>	268.98	268.98	**11.80**
Train FLOPs (P)	131.32	<u>69.16</u>	152.30	69.39	159.26	125.46	**67.76**
Answer tok.	29.9	**4.47**	29.9	11.5	6.14	12.4	<u>4.50</u>

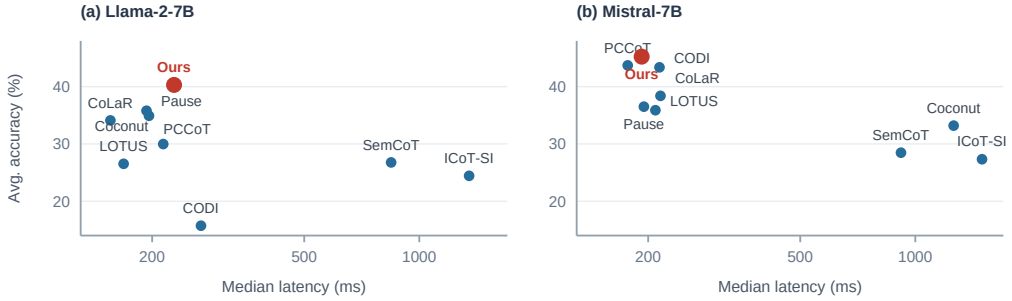


Figure 4: Accuracy–latency comparison for the two reasoners. The horizontal axis denotes median GSM8K latency and the vertical axis the five-task average accuracy, matching Table 3.

tion among the methods reported in Table 3. At inference, its latency remains comparable to the fastest latent baselines despite the additional student extraction and feedback prefill, while retaining a compact answer length. In short, it spends a small trainable interface and one extra prefill to avoid decoding a reasoning trace. These results suggest that the gains are not simply attributable to increased trainable capacity.

5 Conclusion

Latent Recirculation refines a fixed, question-conditioned latent workspace by feeding its final-layer latent states back into the first-layer keys and values through a single shared gate, with a training-only branch supplying directional supervision and both backbones frozen at inference. Across five benchmarks and two reasoners, it attains the best five-task average among the compared latent methods with among the smallest trainable footprint and training computation, and its ablations show that feedback and transition supervision contribute jointly. Overall, the results demonstrate that transition-supervised feedback provides an effective mechanism for refining compact latent reasoning representations without intermediate textual decoding. Because both pretrained backbones remain frozen and only a small latent interface is trained, the approach can be attached to existing reasoners. More broadly, our findings suggest that effective latent recirculation depends on how the workspace update is supervised, rather than on simply repeating the computation, and that a single well-supervised feedback pass is sufficient in the regime we study. We hope this perspective, which treats latent reasoning as a supervised refinement of a fixed workspace rather than as deeper recurrence, informs future designs that revisit the shared-update assumption with depth-aware mechanisms.

Reproducibility Statement

We provide detailed training and inference schedules, gradient boundaries, model and adapter configurations, data processing and answer extraction protocols, baseline hyperparameters, and efficiency measurement settings in Appendices A and B. Task-level ablations, depth results, and sensitivity settings are further reported in Appendix C.

AI Use Statement

Generative AI tools assisted with manuscript editing, literature lookup, LaTeX formatting, figure-generation code, and checks of mathematical notation and result interpretation. The authors are responsible for the final text, claims, and reported experiments.

References

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise lora. In *International Conference on Learning Representations*, volume 2025, pp. 34282–34327, 2025.

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, Yejin Choi, and Stuart Shieber. From explicit cot to implicit cot: Learning to internalize cot step by step. *arXiv preprint arXiv:2405.14838*, 2024.

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

Jonas Geiping, Sean 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. In *Advances in Neural Information Processing Systems*, volume 38, pp. 46242–46293. Curran Associates, Inc., 2025. doi: 10.52202/085713-1380.

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.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. In *International Conference on Learning Representations*, volume 2024, pp. 27896–27923, 2024.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

Yinhan He, Wendy Zheng, Yaochen Zhu, Zaiyi Zheng, Lin Su, Sriram Vasudevan, Qi Guo, Liangjie Hong, and Jundong Li. Semcot: Accelerating chain-of-thought reasoning through semantically-aligned implicit tokens. In *Advances in Neural Information Processing Systems*, volume 38, pp. 48482–48512. Curran Associates, Inc., 2025. doi: 10.52202/085713-1445.

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.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. In *International conference on learning representations*, volume 2024, pp. 32808–32824, 2024.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

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

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? In *Proceedings of the 2021 conference of the North American chapter of the association for computational linguistics: human language technologies*, pp. 2080–2094, 2021.

Subhro Roy and Dan Roth. Solving general arithmetic word problems. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pp. 1743–1752, 2015.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations*, volume 2025, pp. 14855–14881, 2025.

Zhenyi Shen, Hanqi Yan, Linhai Zhang, Zhanghao Hu, Yali Du, and Yulan He. Codi: Compressing chain-of-thought into continuous space via self-distillation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 677–693, 2025.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, 2019.

Wenhui Tan, Jiaze Li, Jianzhong Ju, Zhenbo Luo, Ruihua Song, and Jian Luan. Think silently, think fast: Dynamic latent compression of llm reasoning chains. In *Advances in Neural Information Processing Systems*, volume 38, pp. 5220–5242. Curran Associates, Inc., 2025. doi: 10.52202/085713-0164.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

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.

Haoyi Wu, Zhihao Teng, and Kewei Tu. Parallel continuous chain-of-thought with jacobi iteration. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 914–926, 2025.

Kevin Xu and Issei Sato. On expressive power of looped transformers: Theoretical analysis and enhancement via timestep encoding. *arXiv preprint arXiv:2410.01405*, 2024.

Yige Xu, Xu Guo, Zhiwei Zeng, and Chunyan Miao. Softcot: Soft chain-of-thought for efficient reasoning with llms. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 23336–23351, 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, 2024.

Table 4: Forward schedule. The final implicit pass supplies the answer logits during training and the prefix cache during inference. The CoT branch is omitted at inference.

Setting	Implicit passes	Schedule
$R = 0$	1	Ordinary prefix/answer computation; no feedback or alignment.
$R = 1$	2	Initial prefix gives S^0; one feedback pass gives S^1 and answer logits/cache.
$R = 2$	3	Initial prefix gives S^0; feedback prefix gives S^1; final feedback gives S^2 and answer logits/cache.
Training target	+1	One detached pass over $[q; c; Z]$, with feedback disabled.

A Pass Schedule and Gradient Boundaries

A training step proceeds as follows:

1. Compute Z from the question and thought slots, retaining gradients to the projection and new thought embedding but freezing the student parameters.
2. Disable feedback and compute S^T under stop-gradient using the current reasoner and CoT trace.
3. For $R > 0$, compute S^0 and retain its graph. For $r = 1, \ldots, R-1$, construct memory only at latent positions and recompute the prefix with feedback.
4. For the final pass, append the teacher-forced answer suffix and mask all non-answer labels. Extract S^R from latent positions and compute mean answer cross-entropy. Causal masking prevents answer leakage into S^R.
5. Compute Eq. 7 and Eq. 8; update only projection, new thought embedding, and reasoner LoRA parameters. For $R = 0$, optimize the answer loss alone.

At inference, the final prefill includes the answer cue and stores the refreshed KV cache. Decode the first answer token from its last-position logits, and decode subsequent tokens with that cache. Disable the feedback hook during one-token decoding. Decoding uses the cache from the final feedback pass; recomputing an ordinary prefix would discard that intervention.

Adapter semantics. For a frozen projection W, an adapter replaces it with $W + s_{\mathrm{LoRA}} BA$, where A and B are trainable low-rank factors and s_{LoRA} is the scaling coefficient; dropout is applied to the adapter input during training and disabled at inference. Adapters are added only on the reasoner, on its query and value projections; the student is used as-is. The reference hook projects feedback with the base W_K, W_V, while local projections include the enabled LoRA updates. For a value adapter $B_V A_V$, the local branch is $(W_V + s_{\mathrm{LoRA}} B_V A_V)h$ and the feedback branch is $W_V \widetilde{s}$. This distinguishes the feedback projection from the adapted local value projection. Memory is norm-matched before projection, with rotary positioning handled by the ordinary attention module afterward.

Numerical and gradient conventions. The memory activation threshold is $\epsilon_{\mathrm{mem}} = 10^{-6}$ and the cosine denominator threshold is $\epsilon_{\cos} = 10^{-8}$; norms and cosine similarities are computed in float32. The reference direction is detached, while gradients pass through the implicit state transitions to the trainable parameter groups. Repeated passes with feedback and dropout disabled yield identical states up to numerical precision. For gradient checkpointing, backward recomputation must use the same feedback memory as the corresponding forward pass.

B Configuration and Experimental Protocol

Table 5 summarizes the method configuration. The student backbone remains frozen throughout training.

Parameter accounting. Let L be the number of reasoner layers, d its hidden size, d_{kv} the total key/value width, and k the LoRA rank. Query and value adapters contribute $Lk(3d + d_{kv})$ trainable parameters when these dimensions are constant across layers. Writing N_P for the actual projection parameter count gives

$$N_{\mathrm{train}} = N_P + d_s + Lk(3d + d_{kv}). \tag{9}$$

Table 5: Method and adapter configuration.

Item	Setting
Frozen parameters	Student pretrained backbone; reasoner pretrained backbone
Trainable parameters	Thought Projection, new thought embedding, reasoner-side LoRA
Latent slots / feedback passes	$m = 12$ / $R = 1$
Feedback	Final hidden state to first-layer latent KVs; base-weight memory projection
Mixing	Single scalar ρ shared across positions and passes; value selected on validation data
Supervision	Final-layer directional loss and mean answer cross-entropy; $\alpha = 0.25$
Reasoner adapters	Query/value projections; dropout 0.05
Target branch	Same current reasoner; no feedback; no gradient; training only
Depth accounting	$R + 1$ implicit passes for $R > 0$; one ordinary pass for $R = 0$

Here d_s accounts for the new thought embedding. For an affine projection, $N_P = d_s d + d$, and for a bias-free linear projection, $N_P = d_s d$. We report the instantiated count rather than assuming a projection architecture. Frozen parameters still contribute to inference memory.

Backbones and checkpoints. For Llama-based experiments, we use Llama-2-7B-Chat as the reasoner and Sheared-LLaMA-1.3B as the frozen student. For Mistral-based experiments, we use Mistral-7B-Instruct-v0.2 as the reasoner and a 1.1B Mistral student. Both backbones remain frozen; only the latent interface and reasoner adapters are trained.

Thought Projection. P_ψ is implemented as a single affine linear layer from the student hidden dimension to the reasoner hidden dimension ($2048 \rightarrow 4096$ for both backbone pairs), with a bias term and without additional activation, normalization, or dropout.

Optimization. We use AdamW with a learning rate of 5×10^{-5}, batch size 4, a maximum sequence length of 512, gradient clipping at 1.0, and a maximum budget of 10 epochs with early stopping (patience 3).

Direct-alignment ablation. For the direct-alignment ablation, we replace transition alignment with a cosine state-matching objective,

$$\mathcal{L}_{\text{direct}} = 1 - \frac{1}{m} \sum_i \cos\big(S_i^R, S_i^T\big)\,, \tag{10}$$

averaged over latent positions with the target S^T detached, while keeping the same loss weight $\alpha = 0.25$ and all other settings unchanged.

Data and answer extraction. We follow the data pipeline of the SemCoT release (He et al., 2025), including the benchmark versions, training and evaluation splits, prompt templates, and reasoner/student pairs. For evaluation, we use a unified final-answer scorer across all methods. Rather than searching the entire generated sequence for a reference number, option character, or state token, we extract the designated final-answer span and compare it with the reference after normalization. Arithmetic answers are normalized for thousands separators, surrounding whitespace, and equivalent numeric forms using decimal or rational arithmetic; CoinFlip uses an explicit vocabulary of final states. Outputs without a valid final answer, or whose extracted final answer does not match the reference, are counted as incorrect. The same procedure is applied to every method.

Reference trace source. We do not generate additional reference traces for Latent Recirculation. Our CoT-guided target branch uses the reasoning trace associated with each example in the preprocessed benchmark release, without further generation, rewriting, or correctness filtering. The trace is used only by our training-time target branch and is absent from the student input and test-time inference. For baselines that require reasoning traces as part of their original training objective, we use the corresponding traces from the same processed dataset.

A stronger reasoner. As an additional stronger-backbone check, we evaluate Qwen3-8B with Qwen3-1.7B on GSM8K, MultiArith, and SVAMP under the same final-answer evaluation protocol (Table 6).

Table 6: Reasoning accuracy (%) under our controlled final-answer evaluation protocol, with Qwen3-8B as the reasoner and Qwen3-1.7B as the student (mean $\pm$ SD over five runs).

Method	GSM8K	MultiArith	SVAMP
Ours	33.00 ± 1.26	82.89 ± 0.74	72.00 ± 1.38
LOTUS	30.30 ± 1.03	71.00 ± 1.42	70.40 ± 0.58
CoT	29.50 ± 1.67	51.67 ± 1.25	69.20 ± 3.68
Coconut	27.50 ± 2.59	59.89 ± 4.06	71.70 ± 0.93
PCCoT	25.00 ± 1.05	24.00 ± 3.05	64.50 ± 3.39

Latent Recirculation reaches 33.0 on GSM8K, 82.9 on MultiArith, and 72.0 on SVAMP, ahead of LOTUS (30.3, 71.0, 70.4), Coconut (27.5, 59.9, 71.7), PCCoT (25.0, 24.0, 64.5), and explicit CoT (29.5, 51.7, 69.2). These results provide additional evidence that the improvements transfer to a stronger and more recent backbone across the three evaluated arithmetic benchmarks.

Baseline protocol. Each baseline preserves its original training objective and method-specific design choices. We standardize common implementation settings where possible, while retaining the method-specific hyperparameters listed in Table 7. The shared configuration comprises the same reasoner and student backbones and tokenizers, the same training and evaluation splits, the same checkpoint-selection criterion, and the same prompt and input format; training budgets and stage schedules follow the values specified in Table 7, with a common early-stopping criterion where applicable. Where a baseline cannot consume the shared input format directly, we adapt the input format only and keep its training objective unchanged. The reported accuracies use the unified final-answer evaluation protocol described above. We report each method's trainable parameters, training computation, and inference latency in Table 3.

Hyperparameters. Common implementation settings for the baselines include batch size 4 with gradient accumulation, maximum sequence length 512, adapter dropout 0.1, and five independently trained runs per configuration. Method-specific optimization and architectural settings, including the per-method learning rate, weight decay, epoch budget, and stage schedule, are listed in Table 7.

Table 7: Per-method hyperparameters. Shared settings are given in the text. Epoch budgets sum the stage-wise budgets for methods that train in stages; where two values are given they refer to Llama-2-7B and Mistral-7B, in that order.

Method	LR	WD	Epochs	Method-specific values
Pause	5×10^{-5}	0.01	10	—
ICoT-SI	1×10^{-5}	0.01	10	remove_per_epoch $= 8$
Coconut	1×10^{-5}	0.01	9	$c = 2$; stages $= 3$; max contemplation tokens $= 5$
CODI	8×10^{-4} / 1×10^{-4}	0.01 / 0.1	10	—
SoftCoT	1×10^{-4}	0.01	10	—
SemCoT	1×10^{-4}	1×10^{-3}	10	—
CoLaR	5×10^{-6}	1×10^{-5}	10	compression factor 4; latent loss weight 1.0; max reasoning tokens 256
PCCoT	5×10^{-5}	0.01	10	iterations 3
LOTUS	5×10^{-5}	0.01	9	looped iterations 3; injection "add"; stages $= 3$

Efficiency measurement. All efficiency numbers are measured on a single NVIDIA A100-SXM4-40GB GPU (CUDA 12.0, PyTorch 2.2.2, Transformers 4.57.0). Every backbone is loaded in bfloat16, with no quantization. Latency is the median wall-clock time of the answer-generation call over the first 200 examples of the GSM8K test split at batch size one, measured from the start of the forward pass to the last answer token and excluding model loading; the first ten examples are discarded as warm-up. We also report the mean number of generated tokens, which accounts for much of the latency difference.

Latency against explicit CoT. We include explicit CoT as a latency reference only, not as an accuracy-matched comparison: it is evaluated zero-shot, whereas our method is trained on the task with gold CoT supervision, so the two are not directly comparable in accuracy and we do not read the accuracy gap as evidence of quality. For the latency reference, zero-shot explicit CoT decodes a full reasoning trace before answering and is therefore much slower than our prefill-based pipeline,

which adds a bounded number of prefill passes and then decodes only the answer: on GSM8K its median latency is 2498 ms for Llama-2-7B and 2952 ms for Mistral-7B, against 228 ms and 192 ms for our method, a speedup of $11.0\times$ and $15.4\times$.

Trace handling. The explicit branch places the latent block after the training trace and assigns positional indices so that each latent position attends to the retained trace. Space for the latent block is reserved before truncating a long trace. Neither the trace nor the reference answer enters the student input or test-time inference.

Selection and uncertainty. For trained methods, task accuracies are reported as means and standard deviations over five independently trained runs. Checkpoint selection for all trained methods uses validation answer negative log-likelihood. Method-specific hyperparameters follow the configurations reported in Table 7 and are not selected using test-set performance. Each feedback-depth configuration is retrained and evaluated at its training depth.

C Additional Experimental Results

Table 8 reports task-level feedback-depth results, Table 9 lists the sensitivity grids, and Table 10 gives the task-level ablations underlying Table 2.

Table 8: Task-level depth results for Llama-2-7B (mean $\pm$ SD accuracy); AVG. is the unweighted five-task mean. Each depth uses a separately trained model.

R	CoinFlip	GSM8K	MultiArith	SVAMP	Common	AVG.
0	54.00 ± 0.55	6.40 ± 1.83	18.44 ± 2.82	44.90 ± 1.46	64.70 ± 0.68	37.69
1	62.20 ± 0.68	6.30 ± 1.44	17.56 ± 2.78	46.20 ± 0.81	69.30 ± 0.87	40.31
2	51.90 ± 3.09	6.10 ± 0.97	9.11 ± 2.15	42.50 ± 2.19	64.90 ± 0.97	34.90

Table 9: Sensitivity grids and fixed settings. Each setting is trained independently. Aggregate accuracies are shown in Figure 3.

Factor	Values	Controlled settings
α	$0.125, 0.25, 0.375, 0.5, 0.75$	$m = 12$, $R = 1$, fixed ρ
m	$5, 8, 12, 16, 18, 20, 22$	$R = 1$, $\alpha = 0.25$, fixed ρ

Table 10: Task-level ablation results. Each entry is mean $\pm$ SD accuracy; configurations match Table 2. Task-level changes need not follow the five-task average.

Configuration	CoinFlip	GSM8K	MultiArith	SVAMP	Common
Llama-2-7B					
Full method	62.20 ± 0.68	6.30 ± 1.44	17.56 ± 2.78	46.20 ± 0.81	69.30 ± 0.87
No feedback	54.00 ± 0.55	6.40 ± 1.83	18.44 ± 2.82	44.90 ± 1.46	64.70 ± 0.68
No alignment	57.70 ± 0.76	8.00 ± 1.94	11.56 ± 1.73	45.10 ± 1.52	61.00 ± 3.32
Direct CoT alignment	59.60 ± 0.65	6.80 ± 1.44	17.22 ± 1.62	44.30 ± 1.25	61.20 ± 3.01
No reasoner LoRA	48.90 ± 0.22	7.80 ± 1.15	12.89 ± 3.50	43.80 ± 2.56	65.80 ± 2.64
No norm matching	66.30 ± 0.57	8.50 ± 1.41	11.11 ± 1.88	44.90 ± 1.43	61.60 ± 2.16
Mistral-7B					
Full method	52.10 ± 2.33	11.30 ± 1.47	29.67 ± 2.55	55.70 ± 2.36	77.50 ± 1.64
No feedback	53.60 ± 1.92	12.10 ± 1.93	26.56 ± 3.76	56.10 ± 1.11	72.30 ± 0.60
No alignment	43.40 ± 1.19	9.60 ± 2.92	23.89 ± 1.62	50.50 ± 1.17	71.20 ± 3.11
Direct CoT alignment	53.00 ± 0.35	9.70 ± 2.56	24.44 ± 1.36	53.60 ± 1.92	70.90 ± 2.66
No reasoner LoRA	57.10 ± 0.22	8.90 ± 2.46	26.22 ± 1.86	55.50 ± 1.22	71.80 ± 1.89
No norm matching	53.50 ± 0.79	8.40 ± 2.51	3.78 ± 0.91	51.10 ± 0.65	72.70 ± 1.25