

Do Latent-CoT Models Think Step-by-Step? A Mechanistic Study on Sequential Reasoning Tasks

Jia Liang¹ Liangming Pan^{2,3}

 jialiang2015@gmail.com  Latent-CoT-Thinking

Abstract

Latent Chain-of-Thought (*Latent-CoT*) aims to enable step-by-step computation without emitting long rationales, yet its mechanisms remain unclear. We study CODI, a continuous-thought teacher–student distillation model, on strictly sequential polynomial-iteration tasks. Using logit-lens decoding, linear probes, attention analysis, and activation patching, we localize intermediate-state representations and trace their routing to the final readout. On two- and three-hop tasks, CODI forms the full set of bridge states that become decodable across latent-thought positions, while the final input follows a separate near-direct route; predictions arise via late fusion at the end-of-thought boundary. For longer hop lengths, CODI does not reliably execute a full latent rollout, instead exhibiting a *partial latent reasoning path* that concentrates on late intermediates and fuses them with the last input at the answer readout position. Ablations show that this partial pathway can collapse under regime shifts, including harder optimization. Overall, we delineate when CODI-style latent-CoT yields faithful iterative computation versus compressed or shortcut strategies, and highlight challenges in designing robust latent-CoT objectives for sequential reasoning.

1. Introduction

Recent *Large Language Models* (LLMs) demonstrate substantial competence on multi-step reasoning tasks, producing correct outputs by integrating information across a sequence of intermediate deductions (Jaech et al., 2024; Guo

¹Institute for Computational and Mathematical Engineering, Stanford University, Stanford, USA ²MOE Key Lab of Computational Linguistics, Peking University, Beijing, China ³Beijing Academy of Artificial Intelligence, Beijing, China. Correspondence to: Liangming Pan <liangmingpan@pku.edu.cn>.

Preprint. February 3, 2026.

et al., 2025a). These capabilities are typically realized through two paradigms. In *implicit reasoning*, the model carries out multi-hop inference internally within its hidden activations, without emitting intermediate steps. In *explicit reasoning*, the model verbalizes intermediate computations as discrete tokens, most commonly in the form of *Chain-of-Thought* (CoT) (Wei et al., 2022) traces.

Recently, a third direction—*latent CoT reasoning* (Zhu et al., 2025b; Deng et al., 2024; Hao et al., 2024; Shen et al., 2025)—has begun to bridge these paradigms. Rather than relying solely on the fixed computational depth of a standard transformer (as in typical implicit reasoning), latent CoT architectures add extra internal compute through mechanisms such as continuous “thought tokens”, iterative refinement, or recurrent state updates. These behaviors are often learned by distilling explicit CoT traces into hidden-state trajectories. In principle, latent CoT offers the benefits of explicit step-by-step computation—greater effective depth and expressivity—while avoiding the decoding overhead and brittleness of generating long natural-language rationales.

However, since latent CoT obscures the reasoning process within high-dimensional continuous vectors, this opacity creates a critical gap in our understanding: without the window of interpretability provided by text, it is difficult to verify whether the model is genuinely reasoning or merely employing sophisticated heuristics. While most mechanistic interpretability work has focused on either explicit traces or implicit reasoning (Cabannes et al., 2024; Zhang et al., 2025a; Biran et al., 2024; Wang et al., 2024a), relatively few studies directly investigate the mechanisms of latent chain-of-thought. Yet understanding these mechanisms is crucial: accuracy alone cannot distinguish multi-step computation from shortcuts, nor can it reveal when extra latent compute successfully increases compositional generalization. Many core mechanistic questions remain, such as when a latent-CoT model forms true intermediate states, where they are represented, and how they are stored, propagated, and used to produce the final answer.

In this paper, we give a mechanistic account of how latent CoT solves sequential multi-hop algorithmic tasks when

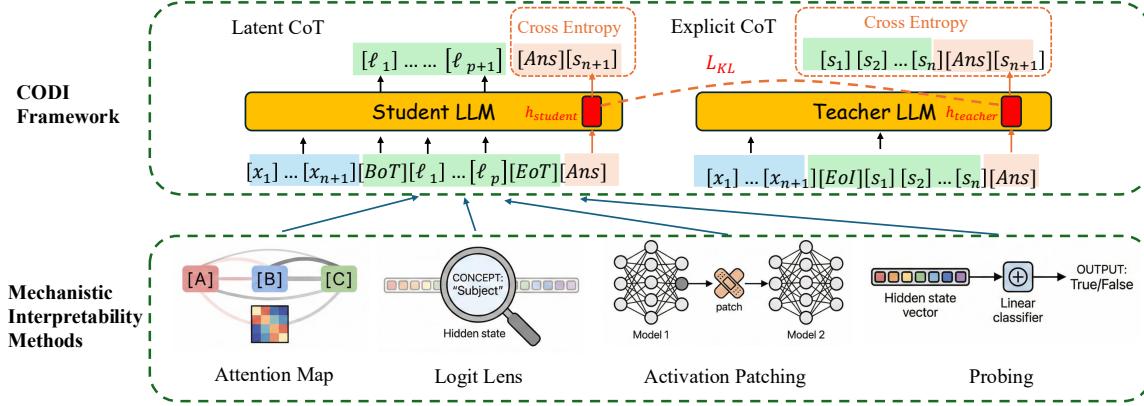


Figure 1. Mechanistic study of CODI on sequential reasoning tasks. *Top:* the CODI training setup used for our polynomial iteration task. *Bottom:* the four mechanistic interpretability methods we use to analyze the student model’s internal computations.

computation is routed through a latent “thought” channel. Using logit-lens (nostalggebraist, 2020) decoding, linear probes (Alain & Bengio, 2016), attention analysis (Vaswani et al., 2017), and targeted activation patching (Meng et al., 2022), we test whether distilling explicit CoT traces into hidden states actually leads the model to internalize genuine step-by-step reasoning, or if it instead relies on shortcuts and late-stage fusion.

Specifically, we use the polynomial sequential tasks introduced by Cabannes et al. (2024) as a controlled testbed to probe latent CoT mechanisms in *Continuous Chain-of-Thought via Self-Distillation (CODI)* (Shen et al., 2025), a typical latent CoT model that uses teacher–student distillation to learn latent reasoning (Figure 1). Polynomial sequential tasks are an ideal testbed for studying CODI because they enforce a transparent, strictly sequential state update with ground-truth intermediates, letting us precisely probe whether and where CODI represents and propagates step-by-step computation within its latent thought channel.

Empirically, our mechanistic analyses reveal a consistent two-route computation in CODI: on 2–3 hop tasks, the latent channel constructs intermediate “bridge” states, while the final input is often delivered to the answer readout via a direct, copy-like pathway. As depth increases ($n \geq 4$), CODI rarely realizes a full latent rollout; instead, the latent channel collapses into a partial-reasoning, late-bottleneck trace that retains only the final one or two intermediates.

Our polynomial-iteration tasks operate over integers modulo m . We also observe a sharp prime–composite split: these mechanistic signatures persist across many composite moduli but largely vanish for prime moduli, where accuracy drops and intermediate-state decodability disappears. This empirical discontinuity motivates our theoretical explanation via *compressibility*: in composite rings, some update steps are inherently many-to-one, which can erase informa-

tion about early inputs and bias the final answer toward a short terminal suffix, making late-bottleneck strategies viable. In contrast, under prime moduli the updates are permutations for nonzero multipliers, so the final answer typically retains genuine dependence on the full history, limiting the benefits of teacher-guided compression and destabilizing step-by-step latent traces.

Comparisons to standard non-CoT transformers and targeted loss ablations suggest that *teacher-guided compression*—distilling explicit-CoT supervision into a short latent trace—drives the partial-reasoning, late-bottleneck strategy under composite moduli. More broadly, relative to fully explicit CoT, latent CoT appears most effective when the underlying computation is *compressible*—as in the composite-modulus regime, where many-to-one contractions reduce the informativeness of early history. This split also highlights a limitation: when updates preserve full-history dependence (as under prime moduli), latent rollouts (step-by-step latent updates) often fail to stabilize.

2. Related Work

To understand how LLMs solve multi-step reasoning problems, a growing line of work applies mechanistic interpretability. In the implicit-reasoning regime, recent studies argue that layers assume distinct computational roles during multi-hop inference (Biran et al., 2024; Li et al., 2024; Yu et al., 2025; Yang et al., 2025c) and identify fine-grained internal structures that support reasoning (Hou et al., 2023; Brinkmann et al., 2024). Other work reports sharp training-phase transitions in which reasoning-like behavior emerges abruptly rather than gradually (Wang et al., 2024a; Ye et al., 2025; Zhang et al., 2025b). Despite this progress, implicit reasoning can be brittle (Biran et al., 2024; Li et al., 2024) and susceptible to shortcut solutions (Ju et al., 2024; Yang et al., 2025b). Several works further suggest that limited

depth is a primary bottleneck, clarifying when and why reasoning fails (Merrill & Sabharwal, 2023; Yu, 2024; Guo et al., 2025b; Saunshi et al., 2025).

On the explicit reasoning side, (Cabannes et al., 2024) and (Dutta et al., 2024) identify attention heads that reuse prior CoT tokens to propagate intermediate results, effectively leveraging generated text as an external memory, while complementary evidence from (Zhang et al., 2025a) and (Rai & Yao, 2024) indicates that CoT models also maintain and update internal state via circuits or neuron activations that encode intermediate variables. Moreover, many more works are investigating *when* (Sprague et al., 2024; Suzgun et al., 2023) and *why* (Li et al., 2023; Yang et al., 2025a) CoT enhance reasoning abilities, as well as its faithfulness (Kudo et al., 2024; Arcuschin et al., 2025).

While prior work shows that latent-CoT representations can be steered or decoded (Zhang & Viteri, 2024; Wang et al., 2024b), these results are primarily correlational and do not directly reveal the underlying mechanism. Building on *COCONUT* (Hao et al., 2024), Zhu et al. (2025a) identify conditions—such as representation superposition—under which latent CoT can outperform explicit CoT on graph-style reasoning. In contrast, we provide a mechanistic account of a different latent-CoT model, *CODI*. Using strictly sequential tasks with ground-truth intermediates, we apply interpretability tools and causal interventions to localize intermediate-state representations and show that CODI-style latent CoT can fail to sustain step-by-step reasoning under specific regimes.

3. Approach

We train CODI on polynomial-iteration tasks with an explicit-CoT teacher and a latent-thought student, using feature-space distillation at the pre-answer [Ans] boundary to align internal states without emitting text. We then analyze the student’s computation with mechanistic interpretability tools to localize intermediate-state representations and distinguish iterative latent updates from shortcut routing (see Figure 1).

The CODI Framework. *Continuous Chain-of-Thought via Self-Distillation (CODI)* is a single-stage framework that compresses an explicit CoT into a short sequence of continuous/latent “thought” vectors while retaining CoT-level accuracy. It trains two modes within the same model: a *teacher* that generates an explicit CoT trace (supervised with cross-entropy on the CoT steps and the final answer), and a *student* that produces a fixed number of latent thought vectors between learned [BoT]/[EoT] (beginning/end-of-thought) markers and is trained with cross-entropy on the final answer. The key supervision is *feature-space self-distillation*: instead of matching a textual rationale, CODI

aligns teacher and student hidden representations at a designated pre-answer boundary using an ℓ_1 loss with stop-gradient on the teacher, so the student learns to reproduce the teacher’s CoT-induced internal state shift without emitting the CoT.

Polynomial-Iteration Dataset. We adopt the polynomial-iteration task from the Iteration Head work (Cabannes et al., 2024) as a controlled testbed for mechanistic analysis. For an n -hop task, we sample inputs $x_1, \dots, x_{n+1} \in \mathbb{Z}_m$ and generate states s_1, \dots, s_{n+1} by $s_1 = x_1$ and

$$s_t = s_{t-1}x_t + b \pmod{m}, \quad t = 2, \dots, n+1, \quad (1)$$

with $m = 50$ and $b = 1$ by default. To instantiate CODI on this task, we train the *teacher* to emit an explicit state-by-state trace and introduce an explicit pre-answer boundary token [Ans] immediately before the final state. The *student* follows CODI’s latent-thought protocol and predicts the final answer after this boundary; the exact teacher/student serializations and the distillation point are shown in Figure 2. Training the *student* uses a cross-entropy loss on s_{n+1} , the final answer, and a CODI-style feature distillation loss matching teacher and student representations at [Ans] (stop-gradient on the teacher).

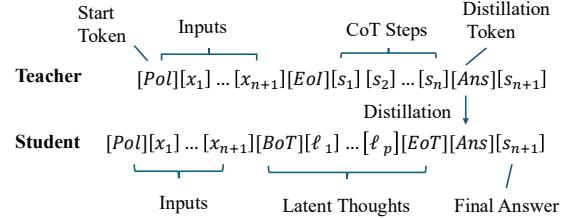


Figure 2. Polynomial tasks for training CODI. The teacher is trained to generate an explicit CoT trace, while the student answers after generating a fixed-length latent-thought trajectory. A feature-space self-distillation loss aligns the teacher and student representations at the answer readout position [Ans].

Mechanistic Interpretability Methods. We use four complementary tools to characterize and localize latent computation in CODI; full method descriptions and implementation details are provided in the referenced appendices. *Logit lens* decodes residual-stream activations (via the unembedding) into a distribution over task states; we apply it across latent positions to track when true intermediates become decodable (Appendix A). *Attention maps* visualize how information is routed across input, latent-thought, and answer positions; we use them to test whether attention implements step-indexed retrieval through the latent tokens (Appendix B). *Linear probes* fit simple classifiers/regressors on hidden states; we train them to predict intermediates (e.g., s_t) at each layer/position to locate where state information is represented and whether the latent trajectory forms an accumulator-like trace (Appendix C). Finally, *activation*

patching is a causal intervention that swaps clean activations into corrupted runs; we patch inputs, latent thoughts, and boundary positions to identify which locations are necessary to recover the correct answer and thus which pathway CODI relies on (Appendix D).

4. Empirical Experiments

A core objective of this work is to test whether *latent* chain-of-thought (CoT) implements genuinely sequential, step-by-step computation, as opposed to producing correct outputs via shallow heuristics. We operationalize this question through the presence of a *bridge state*: an internal representation of the intermediate variable that must be computed after the first hop and then used to complete the second hop. We begin with the simplest controlled setting—two-hop instances in the polynomial task with sequence length three. The default CODI model is a three-layer, two-head GPT-2-style transformer (see Appendix E for full training details).

4.1. Do bridge states form, and where are they represented?

To determine *whether and where* the bridge state forms, we apply the *logit lens* to the residual stream at every latent step and token position and track the decodability of the intermediate tokens s_1, s_2, s_3 . In the two-hop version of our polynomial task, the inputs are x_1, x_2, x_3 , with

$$\begin{aligned} s_1 &= x_1, \quad s_2 = s_1 x_2 + 1 \pmod{50}, \\ s_3 &= s_2 x_3 + 1 \pmod{50}, \end{aligned}$$

where s_3 is the final answer. The central question is whether an explicit representation of the bridge state s_2 becomes decodable *before* the model outputs the final answer.

As shown in Figure 3, the *logit lens* indicates that the token corresponding to s_2 is decodable throughout latent steps $[\ell_1] - [\ell_6]$, with mean decoded probability ranging from 0.359 to 0.709 (averaged over layers and test inputs). We observe the same trend with linear probing (Figure 9 in Appendix C): an s_2 signal emerges after layer 2 at the [BoT] position and remains decodable across all six latent steps, with probe confidence approaching 1. Together, these results suggest that the model uses the latent channel to *instantiate and maintain* an internal representation of the bridge state s_2 prior to producing the final answer.

4.2. How is the final answer computed, and how does information flow to [Ans]?

Having established that the bridge state s_2 becomes decodable over the latent trajectory, we next ask how this intermediate representation is *used* to produce the final answer: what routes information into the final readout position, and

how are the two required inputs (s_2 and x_3) combined? We analyze attention map across layers to identify the dominant sources contributing to its residual stream.

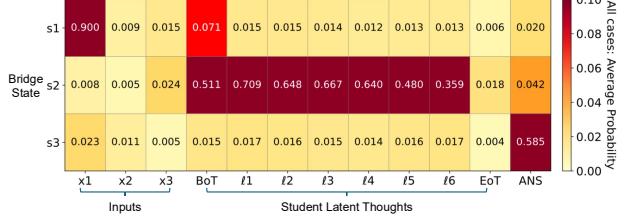


Figure 3. Logit Lens on intermediate states s_1, s_2, s_3 in the two-hop polynomial task. The bridge state s_2 becomes decodable during the latent computation, indicating that it is formed and maintained in the model’s latent channel. Each cell show average decoding probability across all layers and all test inputs.

Attention Map. From the attention maps in Figure 8 (Appendix B), we observe several heads in which the [EoT] and [Ans] tokens place high attention weight directly on the x_3 token (e.g., Head 1/2 in Layers 1–2, and Head 1 in Layer 3). This pattern is consistent with *a copy-like pathway that routes information from x_3 into the residual streams of [EoT] and [Ans]*. This interpretation is supported by linear probing Figure 10 (Appendix C): x_3 is not decodable throughout the latent steps $[\ell_1] - [\ell_6]$, but becomes strongly decodable at [EoT] and [Ans], with probe probability near 1 at [EoT] and approximately 0.85 at [Ans].

Combining this with the logit-lens and probing results indicating that the bridge-state representation (e.g., s_2) is maintained across latent steps, we see a clear division of labor: x_3 is delivered through an early, skip-like pathway, while the intermediate state is stored and updated within the latent computation and only incorporated later into [Ans]. The final prediction is then produced by mixing these two information streams in the [Ans] residual stream prior to unembedding.

Causal Evidence from Interventions. Activation patching provides complementary causal evidence for a direct $x_3 \rightarrow$ [Ans] routing pathway. From Figure 4b, patching the residual stream from a correct run into an x_3 -corrupted run at the latent-step positions ($[\ell_1] - [\ell_6]$, across layers) yields no accuracy recovery (0% average over all corrupted samples), indicating that the latent computation does not carry the x_3 signal in a way that affects the output. In contrast, applying the same patch at the [Ans] position restores performance: for the 3-layer transformer, injecting the patch after the second layer (L2-Post in Figure 4b) recovers 100% accuracy.

Moreover, Figure 4a shows that patching the residual stream from a correct run into an x_2 -corrupted run at the *start of the latent computation* yields substantial recovery. The

effect is strongest at the $[B\circ T]$ position and remains large for early latent steps ℓ_1 and ℓ_2 . In particular, patching at L2-Post on the $[B\circ T]$ token restores accuracy to 100%; recovery is still around 80% for ℓ_1 across layers and around 70% for ℓ_2 in earlier layers. Strikingly, patching at later latent steps produces no recovery (0%), suggesting that the causally relevant computation for this task occurs early in the latent trajectory. These results have two implications. First, $[B\circ T]$ is not merely a delimiter that marks the onset of latent reasoning—it participates in the computation and can carry causally important state. Second, for this two-hop polynomial task, a 6-step latent trajectory appears longer than necessary: mechanistic interventions can therefore provide practical guidance for selecting the number of latent steps in CODI-like models.

Together, these causal effects indicate that the intermediate state is constructed and used at $[B\circ T]$ and early latent positions in a way that directly impacts the final output. We observe the same qualitative pattern when patching x_1 , further strengthening this conclusion. Taken with the decodability and attention results, the evidence is most consistent with a two-stream mechanism: the latent trajectory is primarily used to construct the bridge state s_2 , while x_3 is routed through a largely direct pathway and combined with the latent-state readout at $[Ans]$.

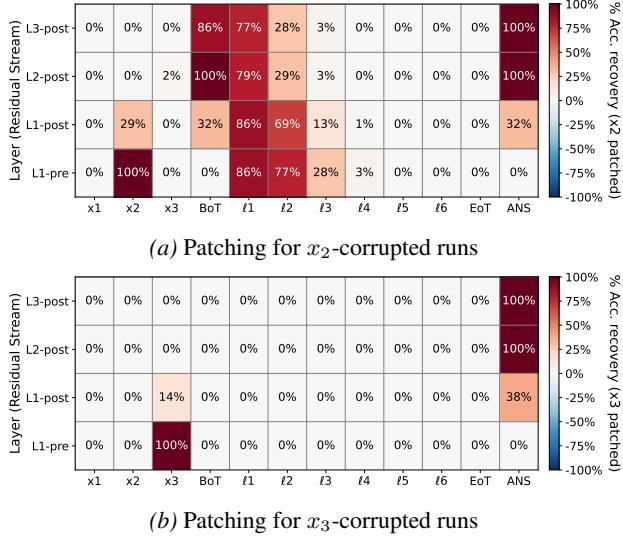


Figure 4. Activation patching for input-token corruptions (two-hop). Cells show mean accuracy recovery (%) over corrupted samples. Patching at latent-thought positions rescues x_2 corruptions, implicating the latent channel in storing s_2 . For x_3 corruptions, recovery concentrates at $[Ans]$, consistent with direct routing of x_3 to the output.

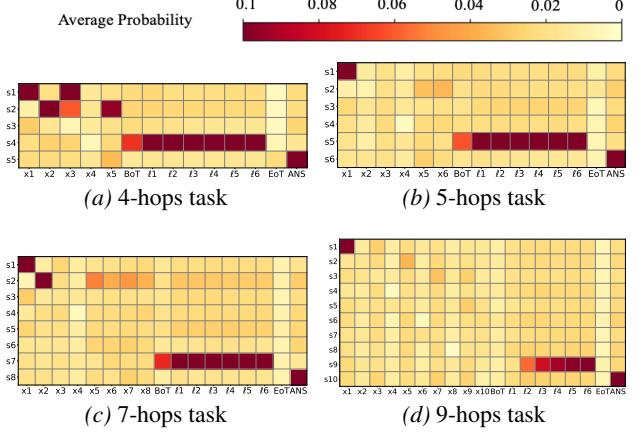


Figure 5. Logit Lens analysis of the n -hop polynomial task with modulus 50. Only the final bridge state s_n appears in the computation pathway, while earlier intermediate bridge states are collapsed.

4.3. How does compositional depth affect the emergent computation?

The two-hop analyses reveal a stable mechanism: CODI (i) forms a bridge state in the latent channel and (ii) fuses it with the final input token at $[Ans]$. We then examine whether greater task depth elicits a longer multi-step latent rollout. To test this, we introduce an n -hop variant with $n \in \{3, 4, 5, 7, 9, 31\}$.

Three-hop Polynomial Task. For the three-hop polynomial task, there are two intermediate bridge states, s_2 and s_3 . We find that CODI exhibits latent reasoning behavior by forming both intermediate states within its latent channel. At the same time, the model appears to use a direct-copy strategy for the final input, with x_4 copied to the $[Ans]$ token to produce the final output. Additional details are provided in Appendix F.

n -hop Polynomial Task, $n \geq 4$. In the n -hop task we have $n+1$ inputs (i.e., x_1, \dots, x_{n+1}) and $n-1$ intermediate bridge states, s_2, \dots, s_n . For $n \geq 4$, increasing hop depth preserves the same two-stream routing pattern but induces a clear *collapse* in intermediate-state visibility. Attention maps continue to reveal an early, direct pathway from the final input x_{n+1} into $[Ans]$ (e.g., strong $[Ans] \rightarrow x_{n+1}$ attention heads in Figure 7). However, logit-lens and probing analyses show that the latent stream does *not* expose a step-by-step chain of intermediates: among s_2, \dots, s_n , only the late intermediate s_n (sometimes the last two intermediates s_{n-1}, s_n as seen in Appendix G) is reliably decodable—both across latent steps $[\ell_1] - [\ell_6]$ and across the input positions x_1, \dots, x_{n+1} , as seen in Figure 5.

Activation patching provides complementary causal evidence for this direct $x_{n+1} \rightarrow [Ans]$ route. When we patch

the residual stream from a clean run into an x_{n+1} -corrupted run at the latent-step positions, accuracy does not recover (0%), indicating that the latent computation does not carry an x_{n+1} signal that causally affects the output. In contrast, applying the same patch at the [Ans] position restores performance in the late layers. Finally, patching a clean run into runs corrupted at earlier inputs x_i (for $i \leq n$) yields nontrivial recovery at specific latent reasoning steps, with larger i generally producing stronger recovery.

It is striking that increasing the nominal hop count does not compel CODI to instantiate an explicit multi-step latent chain that sequentially produces (s_1, \dots, s_n) . Instead, greater depth amplifies a *late-stage bottleneck*: the latent trajectory is dominated by forming and refining a near-final intermediate (here, s_n), while the final input x_{n+1} is routed through a separate, direct pathway. The model then produces the answer via late fusion at [Ans]. We further increase the hop count to $n = 31$, following the same experimental setup as the iteration-head task. Even at this depth, we continue to observe the previously described partial latent pathway.

One plausible hypothesis is a compute limitation: CODI may not have sufficient latent steps to represent and propagate the full sequence of intermediate states. As an alternative, it adopts a *partial* reasoning route that decomposes the computation into two subproblems. In this view, the direct $x_{n+1} \rightarrow [\text{Ans}]$ routing effectively reduces the latent burden to predicting s_n and then combining it with x_{n+1} at [Ans]—a simpler objective than explicitly rolling out the full chain to produce s_{n+1} (i.e., the final answer) entirely within the latent trajectory. This hybrid strategy reduces the difficulty of the latent computation while still retaining enough structure to generalize across hop depth.

4.4. How does task definition affect the observed mechanism?

Our default iteration rule is $s_t = F(s_{t-1}, x_t) = (s_{t-1} \cdot x_t + b) \pmod{m}$, with $b = 1$ and $m = 50$. We evaluate robustness to the task specification by varying the additive constant ($b \in \{1, 3, 4\}$) and the modulus ($m \in \{41, \dots, 50\}$). Changing b does not qualitatively affect the behaviors reported above: across all tested values, we observe the same intermediate-state formation patterns and attention-routing signatures.

However, the choice of m has a much larger effect. Under composite moduli, the mechanistic patterns remain largely unchanged. In contrast, when m is prime, performance drops sharply (Table 1) and the late-intermediate “partial-rollout” signature vanishes (i.e., s_n is no longer decodable during the latent steps as seen in Figure 6). We provide mathematical intuition for this behavior in the next section.

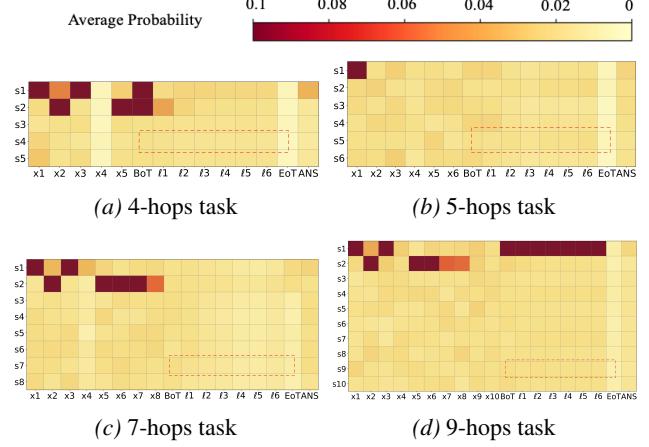


Figure 6. Logit Lens analysis of the n -hop polynomial task with modulus 41. The final bridge state s_n (in dotted rectangular box) is no longer decodable along the latent trajectory, indicating a breakdown of step-by-step reasoning.

5. Theoretical Analysis

Motivated by our empirical finding that CODI collapses—and that the partial, late-bottleneck reasoning signature disappears—when the modulus is prime, we develop a theoretical analysis of the polynomial sequential tasks. We show that the same task family differs substantially in difficulty under prime versus composite moduli.

Task setup. Fix a modulus $m \geq 2$ and consider the ring $R_m := \mathbb{Z}/m\mathbb{Z}$. Given inputs $x_1, \dots, x_T \in \{1, \dots, m-1\}$, define a sequential state by

$$\begin{aligned} s_1 &:= x_1, \\ s_t &:= f_{x_t}(s_{t-1}), := s_{t-1}x_t + b \pmod{m}, \end{aligned} \quad (2)$$

for $t = 2, \dots, T$, and let the label be $y := s_T \in R_m$. We analyze how the algebraic structure of R_m changes the intrinsic dependency of y on the prefix (x_1, \dots, x_{T-k}) .

Affine maps are permutations iff the multiplier is a unit. Let $f_x(s) = sx + b$ over R_m .

Lemma 5.1 (Bijection criterion). *For $x \in R_m$, the map $f_x : R_m \rightarrow R_m$ is bijective iff x is a unit in R_m , i.e. $\gcd(x, m) = 1$.*

Proof. See Appendix H.1.

Composite moduli create many-to-one “contractions.”

When $\gcd(x, m) > 1$, multiplication by x collapses information.

Lemma 5.2 (Exact contraction factor). *Let $d := \gcd(x, m)$. Then the map $s \mapsto sx \pmod{m}$ has image size m/d , and every output has exactly d preimages. Equivalently, $f_x(s) = sx + b$ is a d -to-1 map.*

Proof. See Appendix H.2.

Table 1. Accuracy (%) of CoT-trained models, a non-CoT standard transformer, and CODI across moduli m . CODI and the non-CoT baseline degrade sharply when m is prime. All results use a 3-layer, 2-head transformer with total sequence length 32, and report accuracy aggregated over all sequence lengths.

Mod m	41	42	43	44	45	46	47	48	49	50
Full-CoT	100.00	100.00	28.66	97.30	100.00	93.60	90.85	100.00	100.00	100.00
Non-CoT	5.44	35.73	5.31	18.44	39.94	5.93	2.08	45.62	8.50	25.81
CODI	8.67	66.40	8.59	34.19	67.05	16.74	8.31	91.07	19.97	43.95

Lemma 5.2 gives a rigorous sense in which certain steps *erase* state information: if $d > 1$, then s_t depends on s_{t-1} only through a coarser equivalence class (a factor- d quotient of the state space).

Prime-field regime implies long-range dependence. Now suppose $m = p$ is prime and inputs are sampled uniformly from $\{1, \dots, p-1\} = \mathbb{F}_p^\times$. Then $\gcd(x_t, p) = 1$ always, so every step f_{x_t} is a permutation of \mathbb{F}_p (Lemma 5.1). Hence the recurrence (2) does *not* contract the state space at any time: information about s_{t-1} is never irreversibly discarded. Unrolling the recurrence yields

$$s_T \equiv x_1 \prod_{i=2}^T x_i + b \sum_{t=2}^T \prod_{i=t+1}^T x_i \pmod{p}, \quad (3)$$

a high-degree polynomial in the inputs over the field \mathbb{F}_p (*Convention*: $\prod_{i=T+1}^T x_i := 1$). Because multiplication by nonzero elements is invertible, there is no algebraic mechanism that systematically annihilates the older product terms in (3); thus $y = s_T$ typically retains genuine dependence on the full history.

Composite-ring regime induces short effective memory under uniform sampling. Now let m be composite and sample $x_t \sim \text{Unif}\{1, \dots, m-1\}$. Define the unit probability

$$\begin{aligned} u(m) &:= \Pr(\gcd(x_t, m) = 1) = \frac{\varphi(m)}{m-1}, \\ q(m) &:= 1 - u(m). \end{aligned} \quad (4)$$

where φ is Euler's totient function. With probability $q(m) > 0$, the final step multiplier is a non-unit and the last update is a many-to-one contraction by a factor $d = \gcd(x_T, m)$ (Lemma 5.2). More generally, let

$$\tau := \max\{t \leq T : \gcd(x_t, m) > 1\}, \quad (5)$$

with the convention $\tau = 0$ if all x_1, \dots, x_T are units. Then the length of the terminal all-unit suffix $L := T - \tau$ satisfies

$$\begin{aligned} \Pr(L \geq k) &= u(m)^k \quad (k = 0, 1, \dots, T), \\ \mathbb{E}[L] &= \sum_{k=1}^T u(m)^k = \frac{u(m)(1 - u(m)^T)}{1 - u(m)}. \end{aligned} \quad (6)$$

Thus, for composite m with $u(m) \ll 1$, there is typically a *recent* contraction event τ close to T . Each such event

shrinks the number of distinguishable states by a factor $d = \gcd(x_\tau, m)$, so the label $y = s_T$ can often be predicted from a low-entropy summary of the past plus only the last few updates (the short suffix after τ).

Implications for learned representations. The above analysis is model-agnostic: it characterizes the dependency structure induced by Eq. (2) under the data distribution. For prime $m = p$, each update is a bijection (Lemma 5.1), so the dynamics never contract; Eq. (3) shows s_T retains genuine dependence on the full history via multiplicative chains of length $\Theta(T)$. Mechanistically, long-horizon (large T) success therefore requires sustained state propagation and routing so the running state can be updated at each step and read out at [Ans]; with limited effective compute (e.g., few latent steps and limited depth/routing capacity), we expect degraded accuracy and missing latent rollouts, consistent with Table 1 and Figure 6. For composite m , non-unit multipliers occur with probability $q(m) > 0$ and induce explicit many-to-one contractions (Lemma 5.2); the trailing unit suffix length L has geometric tails (Eq. (6)), so recent contractions make s_T largely depend on a low-entropy summary of the prefix plus the last few updates. This favors predictors that emphasize late intermediates, explaining why CODI often adopts late-bottleneck, partial (late-only) internal rollouts under composite moduli.

6. Ablation Study

We evaluate robustness on long-horizon instances ($n = 31$) with a 3-layer, 2-head student and $m = 50$. Varying the number of latent steps ($p \in \{1, 2, 3, 6, 9, 12, 20\}$) and sweeping the architecture (2–7 layers, 2–8 heads) does not change our qualitative mechanistic picture: CODI primarily encodes and routes late intermediates (s_n , or both s_{n-1}, s_n) and increasing p does not reliably induce a deeper rollout. In loss ablations, removing feature distillation alone preserves these signatures, whereas removing both distillation and the teacher objective eliminates them, indicating that the teacher loss is the key driver of the late-bottleneck latent mechanism. More details are provided Appendix I.

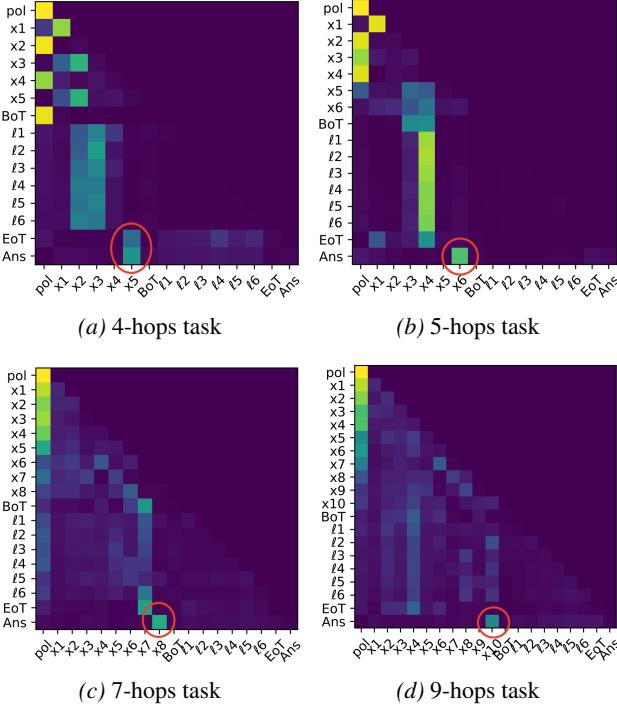


Figure 7. Specialized attention head on the n -hop polynomial task. The $[\text{Ans}]$ token attends strongly to the final input token, x_{n+1} , suggesting a generalized pathway in which x_{n+1} is directly routed to $[\text{Ans}]$.

7. Discussion

Comparison with Non-CoT. A standard next-token transformer trained *without* an explicit or latent CoT channel can nevertheless learn a full internal rollout on this task. In the three-layer, two-head model, the intermediate state s_i becomes decodable at the next input position x_{i+1} (Appendix J, Figure 14a), consistent with an implicit iterative update carried in the residual stream despite supervision only on the final answer. This step-by-step trace is *brittle*, however: it can vanish under modest architectural or optimization changes (Figure 14b), where intermediate-state decodability disappears in a five-layer, two-head model. By contrast, CODI under composite moduli exhibits a *late-bottleneck* behavior on long n -hop tasks: typically only s_n —and occasionally (s_{n-1}, s_n) —becomes decodable in the final latent steps. We attribute this to *teacher-guided compression*: explicit trace supervision pushes the student to compress multi-step computation into a fixed latent-thought budget, and when the task permits short effective memory (Section 5), this pressure favors late-only rollouts.

Compare with CoT. In contrast to both the non-CoT baseline and CODI-style latent CoT, an explicitly CoT-trained transformer can reliably recover the full intermediate trajectory s_1, \dots, s_n across model scales and optimization settings (Cabannes et al., 2024). Consistent with this, Ta-

ble 1 shows that CoT training substantially outperforms both CODI and a standard non-CoT transformer when the modulus m is prime. These results expose a key limitation of CODI-style latent CoT on strictly sequential tasks: with a fixed latent-compute budget, CODI can struggle when the underlying computation is effectively incompressible. This mirrors prior observations that implicit reasoning in standard transformers is constrained by model depth; latent-CoT models inherit a similar constraint, now jointly bounded by architectural depth and the number of latent-thought steps. Conversely, CODI recovers much of the performance gap in more compressible regimes, such as composite moduli.

Composite Rings as a Mechanistic Lens on Latent Reasoning Compression. Studying composite moduli is valuable because their built-in d -to-1 contractions offer a clean analogue of a common feature of language: distinct underlying interpretations can map to similar surface forms. Such ambiguity can reduce the effective value of preserving fine-grained early context once later cues dominate. Composite moduli make this explicit: when an update uses a non-unit multiplier, it collapses d distinct states into one, inducing genuine information loss and shifting the final answer toward dependence on a short terminal suffix. This perspective clarifies when latent CoT is advantageous: in settings with many-to-one mapping from histories (or hidden states) to the observed target/label, additional internal steps may be better used to *compress* history than to preserve it exactly, and teacher-guided latent CoT can act as a structured bottleneck that distills multi-step traces into a small number of latent updates that retain only the intermediates most predictive of the answer (in our polynomial setting, typically the final one or two). We view this as a hypothesis for how latent CoT may help in more naturalistic settings.

8. Conclusion

We mechanistically analyzed CODI on polynomial-iteration tasks. With composite moduli, CODI is step-by-step on 2–3 hops but shifts to a late-intermediate partial rollout on longer horizons; with prime moduli, it largely fails to learn and shows neither signature. Our theory explains this split via *compressibility*: composite moduli induce many-to-one contractions that bias the label toward late updates, making the computation effectively compressible and enabling CODI to succeed with a fixed latent budget. Prime-modulus instances lack this structure and are substantially less compressible. These findings clarify when CODI-style latent CoT yields faithful iterative computation versus compressed late-stage solutions, and they highlight a key failure mode on sequential tasks that cannot be stably compressed.

Looking ahead, a natural next step is to test whether the compressibility dependence we observe (including the prime–composite split) persists across latent-CoT objectives and

architectures, and to develop adaptive mechanisms that allocate latent compute to match task demands. Extending our mechanistic toolbox to more sequential and naturalistic datasets may further clarify when latent reasoning implements faithful multi-step computation versus shortcut- or compression-driven strategies.

Impact Statement

Mechanistic interpretability aims to understand the computations implemented by deep learning systems by linking behavior to internal representations and circuits. Because latent chain-of-thought moves reasoning into hidden activations, mechanistic study can improve safety and reliability by revealing when models carry out faithful multi-step computation versus shortcut or compression-driven strategies, and by informing the design of more predictable objectives. At the same time, such understanding could also enable more capable models by making internal computation more efficient, potentially amplifying broader societal issues associated with advanced AI systems; these issues are beyond the scope of this statement. Our experiments are conducted in a deliberately controlled synthetic setting (polynomial-iteration tasks over modular arithmetic) with randomly sampled inputs and ground-truth intermediate states; the data contain no real-world entities or personal information, and no human subjects are involved, so we do not anticipate privacy, consent, or representational harms originating from the dataset.

References

- Alain, G. and Bengio, Y. Understanding intermediate layers using linear classifier probes. *arXiv preprint arXiv:1610.01644*, 2016.
- Arcuschin, I., Janiak, J., Krzyzanowski, R., Rajamanoharan, S., Nanda, N., and Conmy, A. Chain-of-thought reasoning in the wild is not always faithful. *URL https://arxiv.org/abs/2503.08679*, 2025.
- Biran, E., Gottesman, D., Yang, S., Geva, M., and Globerson, A. Hopping too late: Exploring the limitations of large language models on multi-hop queries. *arXiv preprint arXiv:2406.12775*, 2024.
- Brinkmann, J., Sheshadri, A., Levoso, V., Swoboda, P., and Bartelt, C. A mechanistic analysis of a transformer trained on a symbolic multi-step reasoning task. *arXiv preprint arXiv:2402.11917*, 2024.
- Cabannes, V., Arnal, C., Bouaziz, W., Yang, X., Charton, F., and Kempe, J. Iteration head: A mechanistic study of chain-of-thought. *Advances in Neural Information Processing Systems*, 37:109101–109122, 2024.
- Deng, Y., Choi, Y., and Shieber, S. From explicit cot to implicit cot: Learning to internalize cot step by step. *arXiv preprint arXiv:2405.14838*, 2024.
- Dutta, S., Singh, J., Chakrabarti, S., and Chakraborty, T. How to think step-by-step: A mechanistic understanding of chain-of-thought reasoning. *arXiv preprint arXiv:2402.18312*, 2024.
- Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025a.
- Guo, T., Zhu, H., Zhang, R., Jiao, J., Mei, S., Jordan, M. I., and Russell, S. How do llms perform two-hop reasoning in context? *arXiv preprint arXiv:2502.13913*, 2025b.
- Hao, S., Sukhbaatar, S., Su, D., Li, X., Hu, Z., Weston, J., and Tian, Y. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.
- Hou, Y., Li, J., Fei, Y., Stolfo, A., Zhou, W., Zeng, G., Bosselut, A., and Sachan, M. Towards a mechanistic interpretation of multi-step reasoning capabilities of language models. *arXiv preprint arXiv:2310.14491*, 2023.
- Jaech, A., Kalai, A., Lerer, A., Richardson, A., El-Kishky, A., Low, A., Helyar, A., Madry, A., Beutel, A., Carnegy, A., et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.
- Ju, T., Chen, Y., Yuan, X., Zhang, Z., Du, W., Zheng, Y., and Liu, G. Investigating multi-hop factual shortcuts in knowledge editing of large language models. *arXiv preprint arXiv:2402.11900*, 2024.
- Kingma, D. P. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- Kudo, K., Aoki, Y., Kurabayashi, T., Sone, S., Taniguchi, M., Brassard, A., Sakaguchi, K., and Inui, K. Think-to-talk or talk-to-think? when llms come up with an answer in multi-step reasoning. *arXiv preprint arXiv:2412.01113*, 2024.
- Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.
- Li, Y., Sreenivasan, K., Giannou, A., Papailiopoulos, D., and Oymak, S. Dissecting chain-of-thought: Compositionality through in-context filtering and learning. *Advances in Neural Information Processing Systems*, 36: 22021–22046, 2023.

- Li, Z., Jiang, G., Xie, H., Song, L., Lian, D., and Wei, Y. Understanding and patching compositional reasoning in llms. *arXiv preprint arXiv:2402.14328*, 2024.
- Meng, K., Bau, D., Andonian, A., and Belinkov, Y. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372, 2022.
- Merrill, W. and Sabharwal, A. The expressive power of transformers with chain of thought. *arXiv preprint arXiv:2310.07923*, 2023.
- Nanda, N. and Bloom, J. Transformerlens. <https://github.com/TransformerLensOrg/TransformerLens>, 2022.
- nostalgiaist. interpreting GPT: the logit lens. LessWrong, 2020. URL <https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens>.
- Rai, D. and Yao, Z. An investigation of neuron activation as a unified lens to explain chain-of-thought eliciting arithmetic reasoning of llms. *arXiv preprint arXiv:2406.12288*, 2024.
- Saunshi, N., Dikkala, N., Li, Z., Kumar, S., and Reddi, S. J. Reasoning with latent thoughts: On the power of looped transformers. *arXiv preprint arXiv:2502.17416*, 2025.
- Shen, Z., Yan, H., Zhang, L., Hu, Z., Du, Y., and He, Y. Codi: Compressing chain-of-thought into continuous space via self-distillation. *arXiv preprint arXiv:2502.21074*, 2025.
- Sprague, Z., Yin, F., Rodriguez, J. D., Jiang, D., Wadhwa, M., Singhal, P., Zhao, X., Ye, X., Mahowald, K., and Durrett, G. To cot or not to cot? chain-of-thought helps mainly on math and symbolic reasoning. *arXiv preprint arXiv:2409.12183*, 2024.
- Suzgun, M., Scales, N., Schärli, N., Gehrmann, S., Tay, Y., Chung, H. W., Chowdhery, A., Le, Q., Chi, E., Zhou, D., et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, 2023.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- Wang, B., Yue, X., Su, Y., and Sun, H. Grokked transformers are implicit reasoners: A mechanistic journey to the edge of generalization. *arXiv preprint arXiv:2405.15071*, 2024a.
- Wang, K. R., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. In *The Eleventh International Conference on Learning Representations*, 2023.
- Wang, Y., Zhang, P., Yang, B., Wong, D. F., and Wang, R. Latent space chain-of-embedding enables output-free llm self-evaluation. *arXiv preprint arXiv:2410.13640*, 2024b.
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
- Yang, C., Li, Z., and Wipf, D. Chain-of-thought provably enables learning the (otherwise) unlearnable. In *The Thirteenth International Conference on Learning Representations*, 2025a.
- Yang, S., Kassner, N., Gribovskaya, E., Riedel, S., and Geva, M. Do large language models perform latent multi-hop reasoning without exploiting shortcuts? In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 3971–3992, 2025b.
- Yang, Z., Li, J., Xia, S., and Hu, X. Internal chain-of-thought: Empirical evidence for layer-wise subtask scheduling in llms. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 22547–22575, 2025c.
- Ye, J., Yao, Z., Huang, Z., Pan, L., Liu, J., Bai, Y., Xin, A., Weichuan, L., Che, X., Hou, L., et al. How do transformers learn implicit reasoning? In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.
- Yu, Y. Do llms really think step-by-step in implicit reasoning? *arXiv preprint arXiv:2411.15862*, 2024.
- Yu, Z., Belinkov, Y., and Ananiadou, S. Back attention: Understanding and enhancing multi-hop reasoning in large language models. *arXiv preprint arXiv:2502.10835*, 2025.
- Zhang, J. and Viteri, S. Uncovering latent chain of thought vectors in language models. *arXiv preprint arXiv:2409.14026*, 2024.
- Zhang, Y., Du, W., Jin, D., Fu, J., and Jin, Z. Finite state automata inside transformers with chain-of-thought: A mechanistic study on state tracking. *arXiv preprint arXiv:2502.20129*, 2025a.

Zhang, Z., Lin, P., Wang, Z., Zhang, Y., and Xu, Z.-Q. J.

Complexity control facilitates reasoning-based compositional generalization in transformers. *arXiv preprint arXiv:2501.08537*, 2025b.

Zhu, H., Hao, S., Hu, Z., Jiao, J., Russell, S., and Tian,

Y. Reasoning by superposition: A theoretical perspective on chain of continuous thought. *arXiv preprint arXiv:2505.12514*, 2025a.

Zhu, R.-J., Peng, T., Cheng, T., Qu, X., Huang, J., Zhu, D.,

Wang, H., Xue, K., Zhang, X., Shan, Y., et al. A survey on latent reasoning. *arXiv preprint arXiv:2507.06203*, 2025b.

A. Logit Lens

Goal. We use the *logit lens* to test whether an internal representation already contains the correct intermediate state for the polynomial iteration task. The logit lens is a lightweight diagnostic: it applies the model’s own output decoder (the *unembedding*) to hidden states from earlier layers and positions, yielding a distribution over discrete tokens/states that can be compared against ground truth. In our setting, this lets us localize *when* the model commits to the correct intermediate value and whether that commitment emerges progressively over latent computation (consistent with iterative updates) or unusually early (suggesting shortcut behavior), which is particularly informative for CODI-style latent reasoning.

Background: output logits. For a decoder-only Transformer with residual-stream vectors $h_t^{(L)} \in \mathbb{R}^d$ at position t after the final layer L , next-token logits are typically computed as

$$z_t = W_U \text{LN}\left(h_t^{(L)}\right) + b_U, \quad p(x_t = v | x_{<t}) = \text{softmax}(z_t)_v, \quad (7)$$

where $W_U \in \mathbb{R}^{|V| \times d}$ is the unembedding matrix, b_U is a bias, $\text{LN}(\cdot)$ denotes the final layer normalization used before decoding, and V is the vocabulary.

Logit lens construction. At a chosen intermediate layer l and token position t (e.g., a pre-answer boundary token or a specific state/thought position), we take the residual stream $h_t^{(l)}$ and map it to a distribution using the same decoder:

$$\tilde{z}_t^{(l)} = W_U \text{LN}\left(h_t^{(l)}\right) + b_U, \quad \tilde{p}_t^{(l)} = \text{softmax}\left(\tilde{z}_t^{(l)}\right). \quad (8)$$

We interpret $\tilde{p}_t^{(l)}$ as: *if the model were forced to decode a token from layer l at position t using its final unembedding (or an equivalent lightweight readout), which discrete value would it predict?*

Metrics & Visualization. For the polynomial sequential task, given the ground-truth intermediate state s^* (e.g., s_t or the final s_n in an n -hop instance), we measure the logit-lens probability assigned to s^* , i.e., $\tilde{p}_t^{(l)}(s^*)$. We plot $\tilde{p}_t^{(l)}(s^*)$ (averaged over examples and all layers) as a function of token position t , including CODI latent-thought positions, to localize *when* the correct intermediate value becomes decodable and to distinguish progressive latent construction from early commitment.

B. Attention Maps

What an attention map shows. An *attention map* visualizes where a self-attention head reads from when updating each token representation. For a fixed layer l and head h , each query position t forms a weighted average of information from (allowed) key/value positions j . The attention map is the matrix of these normalized weights, with rows corresponding to queries t and columns to keys j .

Self-attention mechanics. Let $x_t \in \mathbb{R}^d$ denote the residual-stream vector at position t entering the attention sublayer. A single head constructs

$$q_t = W_Q x_t, \quad k_j = W_K x_j, \quad v_j = W_V x_j, \quad (9)$$

and computes scaled dot-product attention scores followed by a softmax:

$$\alpha_{t \rightarrow j} = \text{softmax}_j\left(\frac{q_t^\top k_j}{\sqrt{d_k}} + m_{tj}\right), \quad (10)$$

where m_{tj} is an attention mask (e.g., a causal mask sets $m_{tj} = -\infty$ for future positions $j > t$). The head output at position t is

$$\text{Attn}(x)_t = \sum_j \alpha_{t \rightarrow j} v_j. \quad (11)$$

Stacking $\alpha_{t \rightarrow j}$ over all t and j yields an attention matrix $A^{(l,h)} \in \mathbb{R}^{T \times T}$, where each row sums to 1.

How we visualize it. We plot $A^{(l,h)}$ as a heatmap: *rows* index query positions t (the positions being updated) and *columns* index key/value positions j (the positions being attended to). Bright entries indicate large $\alpha_{t \rightarrow j}$, i.e., strong routing from position j into the update at position t . In our setting, positions include input tokens, structural boundary tokens (e.g., [BoT], [EoI], [Ans]), and (for CODI) continuous thought tokens ($[\ell_1] \dots [\ell_6]$). Unless otherwise noted, we visualize individual heads and often average $A^{(l,h)}$ over a batch of examples to highlight stable routing structure. We annotate rows/columns by token type (input vs. thought vs. boundary) to clarify information flow.

Figure 8 visualizes attention maps from the student of a 3-layer, 2-head Transformer trained with the CODI objective on the two-hop polynomial task. Several heads exhibit strong attention from the [Ans] position (immediately before answer generation) to the final input token x_3 , consistent with a copy-like pathway that routes x_3 directly into the [Ans] residual stream.

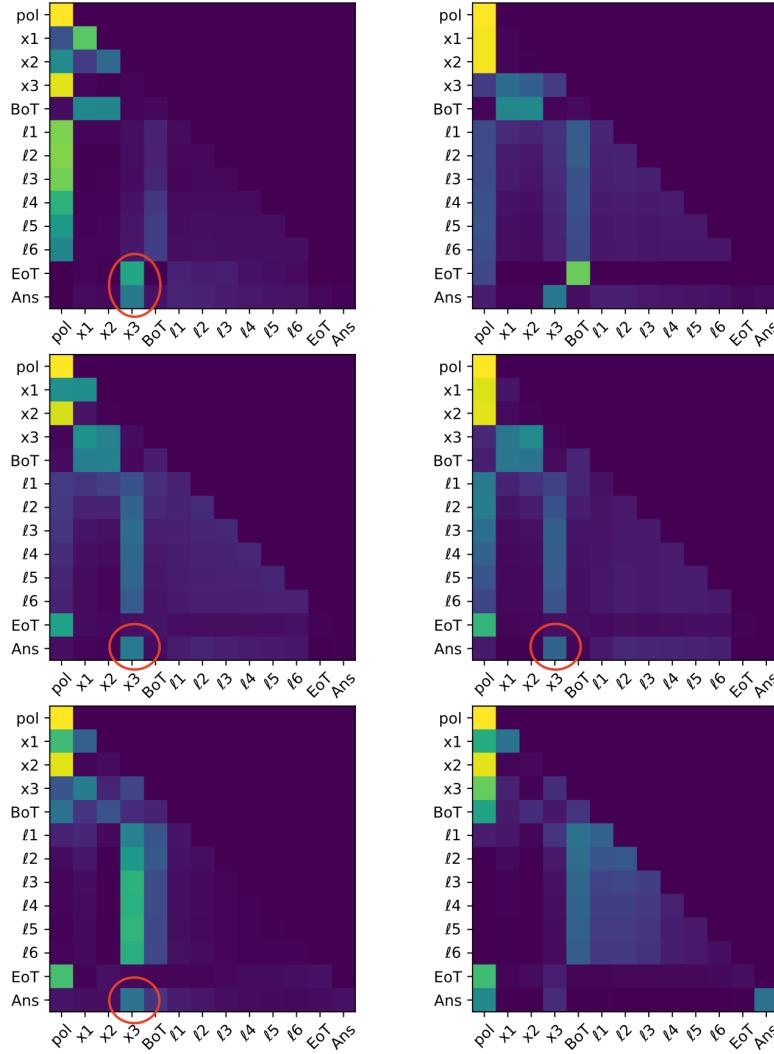


Figure 8. Attention maps for the 3-layer, 2-head transformer on the two-hop polynomial task. Rows correspond to layers (Layer 1 on top through Layer 3 on bottom), and columns correspond to heads (Head 1 on the left, Head 2 on the right). Across multiple heads, the [Ans] token places strong attention directly on the x_3 position (circled in red ovals), consistent with a copy-like pathway that routes x_3 into the [Ans] residual stream.

C. Probing Intermediate Representations

Goal. *Probing* is a diagnostic technique for quantifying what information is present in a model’s hidden states. Given intermediate activations (e.g., residual-stream vectors) produced while the model processes an input, we train a simple supervised predictor (a *probe*) to recover a task-relevant variable from those activations. If a low-capacity probe can accurately predict a variable (e.g., an intermediate state), this provides evidence that the variable is encoded in the representation in an easily accessible (often approximately linear) form.

Representations and targets. For each example, we extract hidden states $h_t^{(l)} \in \mathbb{R}^d$ at layer l and token position t . In the polynomial sequential task, we train probes to predict task-relevant variables, including the current state s_t , the next state s_{t+1} , the final answer s_{n+1} , and (for an n -hop instance) each input token x_1, \dots, x_{n+1} . We evaluate probes across token types/positions, including input tokens, CODI continuous thought tokens, and pre-answer boundary tokens.

C.1. Linear Probing Implementation Details

Overview. We use linear probes to quantify which task variables are linearly decodable from CODI’s internal representations. Probes are trained on frozen activations extracted from runs where the model predicts the correct final answer, and we report held-out classification accuracy.

Probe architecture. Each probe is a single linear classifier with no bias and no nonlinearity:

$$f(h) = Wh, \quad (12)$$

where $h \in \mathbb{R}^{256}$ is the hidden vector and $W \in \mathbb{R}^{50 \times 256}$. The 50 classes correspond to discrete values $\{0, 1, \dots, 49\}$.

Activation extraction (where we probe). For a sequence length n , we probe representations at the following positions:

- **Input positions:** x_i for $i \in \{1, \dots, n+1\}$ (residual stream at each input token)
- **BoT (Beginning of Thought):** final encoder position before latent processing
- **Latent thought tokens:** ℓ_j for $j \in \{1, \dots, 6\}$ (six learned latent tokens)
- **EoT (End of Thought):** first decoder position after latent processing
- **Ans:** decoder position used for answer generation

At each position, we probe the residual stream at four depths: before the first layer (L1-Pre) and after layer 1, 2, 3 (L $\{0, 1, 2\}$ -post).

Probe targets (what we decode). We train separate probes for each of the following ground-truth labels:

- **Inputs:** x_1, \dots, x_n with $x_i \in \{1, \dots, 49\}$
- **Intermediate states:** s_1, \dots, s_{n-1} with $s_i \in \{0, \dots, 49\}$
- **Final answer:** ans $\in \{0, \dots, 49\}$

Dataset and filtering. For each sequence length, we extract activations from approximately 5,000 test examples. To focus on representations associated with successful computation, we retain only examples where the model’s final prediction is correct, yielding N_{correct} samples.

Train/validation/test splits. We split N_{correct} into 80% train and 20% test. The training portion is further split into 80% for optimization and 20% for validation, giving an overall 64%/16%/20% train/val/test split.

Optimization. Each probe is trained independently using Adam (Kingma, 2014) (learning rate 10^{-3}), batch size 64, for 100 epochs, minimizing cross-entropy loss. We select the checkpoint with the highest validation accuracy and report test accuracy for that probe.

Number of probes. For sequence length n , we train probes over: (i) **locations:** n encoder positions + 1 BoT + 6 latents + 1 EoT + 1 Ans = $n + 9$; (ii) **depths:** 4 residual-stream depths; and (iii) **labels:** n inputs + $(n - 1)$ intermediate states + 1 answer = $2n$. This yields $(n + 9) \times 4 \times 2n$ probe fits per sequence length (e.g., 896 probes when $n = 7$).

Metric. We report classification accuracy on the held-out test split. Chance performance is $1/50 = 0.02$; accuracy near 1.0 indicates the target variable is fully linearly decodable from the probed representation.

Probing Visualization. Figure 9 and Figure 10 visualize linear-probe results on the two-hop polynomial task (modulus 50), with probe targets s_2 (the intermediate bridge state) and x_3 (the final input), respectively. Each cell reports classification accuracy on the held-out test split; the x -axis indexes the token positions from which activations are extracted, and the y -axis indexes the four probed residual-stream depths at each position. In Figure 9, s_2 is highly decodable across the latent thought tokens $[\ell_1] - [\ell_6]$, indicating that the latent channel carries the intermediate-state information. In contrast, Figure 10 shows that x_3 is largely not decodable within the latent tokens, but becomes strongly decodable at [EoT] and [Ans]. This pattern is consistent with the copy-like route suggested by the attention maps, where x_3 is routed directly into the final readout.



Figure 9. Probing the intermediate state s_2 in the two-hop polynomial task. Consistent with the logit-lens results, s_2 is readily decodable throughout the latent computation, indicating that the bridge state is formed before the model produces the final answer and supporting a sequential reasoning strategy.

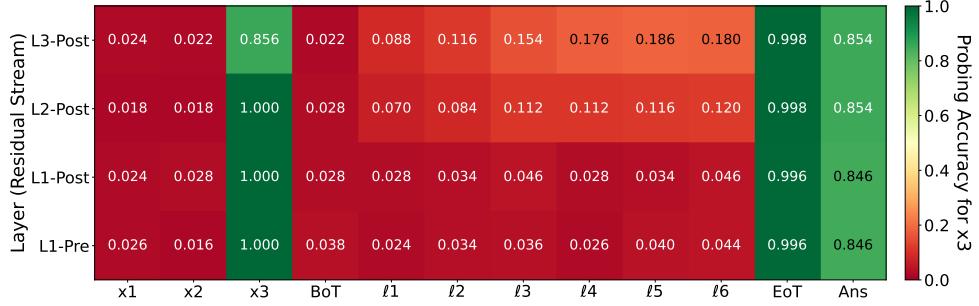


Figure 10. Probing the input token x_3 in the two-hop polynomial task. x_3 is strongly decodable at its own position and at the [EoT] and [Ans] tokens (probe confidence ≈ 1 at x_3 and [EoT], and ≈ 0.85 at [Ans]). Consistent with the attention pattern in Figure 8, this supports a direct routing (copy-like) pathway from x_3 into the [EoT] and [Ans] residual streams.

D. Activation Patching

Goal. *Activation patching* is a causal intervention used to identify which internal components of a model are necessary (or sufficient) for a particular behavior. The core idea is to compare a *clean* run of the model (which produces the correct answer) to a *corrupted* run (where an input is perturbed and the model fails), and then *replace* a chosen internal activation in the corrupted run with the corresponding activation from the clean run. If this replacement restores the correct output, it provides causal evidence that the patched activation carries information that the model needs to solve the task.

D.1. Activation Patching Implementation Details

Overview. We employ activation patching (Meng et al., 2022; Wang et al., 2023), also known as causal tracing or interchange intervention, to identify which model components are causally necessary for correct predictions. Our implementation extends standard activation patching to handle the encoder-latent-decoder architecture with multiple latent reasoning tokens.

Clean and Corrupted Runs. For each test example with input sequence $\mathbf{x} = (x_1, \dots, x_n)$ and correct answer y , we define:

- **Clean run:** Forward pass with correct inputs, storing all intermediate activations $\mathcal{A}^{\text{clean}} = \{a_\ell^{(p)} : \ell \in \mathcal{L}, p \in \mathcal{P}\}$, where \mathcal{L} indexes layers and \mathcal{P} indexes positions/phases.
- **Corrupted run:** Forward pass with perturbed input $\tilde{\mathbf{x}}$, where we corrupt one token: $\tilde{x}_i = (2 \cdot x_i) \bmod 50$ for some position i .

We cache activations from three computation phases: encoder positions $\{1, \dots, n\}$, latent tokens $\{\ell_1, \dots, \ell_6\}$, and decoder positions $\{\text{EoT}, \text{Ans}\}$.

Intervention Procedure. Given corrupted input $\tilde{\mathbf{x}}$ and clean activation cache $\mathcal{A}^{\text{clean}}$, we perform the following intervention at component $c = (\ell, p)$ (layer ℓ , position/phase p):

$$a_\ell^{(p)}(\tilde{\mathbf{x}}) \leftarrow a_\ell^{(p)}(\mathbf{x}) \quad (13)$$

We implement this via forward hooks in TransformerLens (Nanda & Bloom, 2022). For position-specific patching at encoder token i or decoder position j , we patch only the corresponding slice: $a_{\ell,i}^{\text{enc}}(\tilde{\mathbf{x}}) \leftarrow a_{\ell,i}^{\text{enc}}(\mathbf{x})$.

Components Analyzed. We systematically patch residual stream activations at the following components:

- **Layers:** Pre-residual (before layer 1: L1-pre) and post-residual after each layer $k \in \{1, 2, 3\}$ (Lk-post)
- **Phases:** For sequence length n , we test n encoder positions + 1 BoT position + 6 latent tokens + 2 decoder positions, yielding $n + 9$ phases

For a model with $L = 3$ layers and sequence length n , this produces $4 \times (n + 9)$ total intervention experiments per corrupted example.

Metrics. For each intervention, we compute:

$$\text{Baseline} = \mathbb{P}(\hat{y} = y | \tilde{\mathbf{x}}) \quad (\text{no patching}) \quad (14)$$

$$\text{Patched}_c = \mathbb{P}(\hat{y} = y | \tilde{\mathbf{x}}, \text{patch at } c) \quad (15)$$

$$\text{Clean} = \mathbb{P}(\hat{y} = y | \mathbf{x}) \quad (\text{upper bound}) \quad (16)$$

$$\text{Lift}_c = \frac{\text{Patched}_c - \text{Baseline}}{\text{Clean} - \text{Baseline}}. \quad (17)$$

The *lift* metric normalizes the recovery to $[0, 1]$, where $\text{Lift}_c = 1$ indicates complete recovery and $\text{Lift}_c = 0$ indicates no improvement. We report lift as the primary metric, as it is invariant to baseline difficulty and allows comparison across different corruptions.

Evaluation Protocol. We filter test examples to include only those where the model initially predicts correctly ($\mathbb{P}(\hat{y} = y | \mathbf{x}) = 1$), ensuring clean runs provide a valid counterfactual. We corrupt each input position independently, running separate patching experiments for x_1, x_2, \dots, x_n . For each corruption, we compute the mean accuracy and lift over all correctly-predicted examples.

D.2. Visualization Details.

Figures 4a and 4b report activation-patching lift for runs corrupted at x_2 and x_3 , respectively. The x -axis indexes the token position (inputs and latent tokens) where we patch, and the y -axis indexes the probed residual-stream depth. Each cell shows mean percent recovery, $\text{Acc Recovery} = 100\% \cdot \text{Lift}_c$, averaged over clean-correct examples. In Figure 4a, patching into the x_2 -corrupted run yields the largest recovery at latent positions $[\ell_1]$ and $[\ell_2]$, indicating that early latent steps carry causally necessary intermediate information. In contrast, Figure 4b shows negligible recovery from patching at latent positions for the x_3 -corrupted run, while patching at [Ans] produces strong recovery, consistent with a copy-like route from the final input x_3 to the [Ans] readout.

E. Training Configuration

We train a 3-layer transformer with 2 attention heads on the polynomial reasoning dataset. The model uses 6 latent tokens and a 256-dimensional projection layer. We train for 1,000 epochs with batch size 256, using AdamW with learning rate 3×10^{-4} and a cosine annealing schedule. We use a warmup ratio of 0.03, weight decay of 0.1, and gradient clipping with max norm 2.0. The objective combines cross-entropy and distillation losses with equal weights (1.0 each); the distillation loss is normalized by its standard deviation. Each run uses 2,500 examples per sequence length. For an n -hop task, we train on a curriculum of sequence lengths $1, 2, \dots, n+1$ (e.g., 4-hop training includes lengths 1–5). All models are trained in BF16 precision for computational efficiency.

F. Mechanistic Analysis on Three-hop Polynomial task.

From the logit-lens analysis in Figure 12, we observe a clear temporal progression of intermediate-state decodability across latent steps. The first state s_1 becomes decodable early in the latent trajectory (specifically from [BoT] through [L6] in this setting). Later in the trajectory, the second intermediate state s_2 becomes decodable at the [EoT] token. Together, these patterns are consistent with step-by-step computation on the three-hop polynomial task.

Activation patching provides causal support for this interpretation. As shown in Figures 11a and 11b, patching from a correct run into an x_1 -corrupted or x_2 -corrupted run yields substantial accuracy recovery primarily in early latent steps (ℓ_1 and ℓ_2). Because s_2 is computed from x_1 and x_2 , this indicates that the latent representation supporting s_2 is causally relevant in these steps. Moreover, Figure 11c shows that patching x_3 has its strongest effect at [EoT], consistent with s_3 being formed and used late in the trajectory. Overall, both analyses support a sequential, step-by-step reasoning process.

It is important to note that this step-by-step reasoning pattern does not emerge uniformly across all polynomial-task runs. In some cases, the model exhibits a *partial* latent trajectory: only s_3 becomes decodable in the latent steps, without clear evidence of s_2 formation. That said, across the different random seeds and small model architectures we tested, the full step-by-step pattern appears reliably in the majority of settings.

G. Partial Latent Rollouts Concentrate on Late Intermediates for Longer Hops.

As shown in Figure 5, we previously observed that, on n -hop tasks, the final intermediate state s_n often becomes decodable within CODI’s latent-thought positions. For longer horizons (typically $n \geq 4$ under composite moduli), we also observe a stronger *two-step* variant of this behavior: both s_{n-1} and s_n become decodable across the latent trajectory, with s_{n-1} emerging earlier and s_n emerging later.

Figure 13a illustrates this pattern on a 5-hop task: s_4 is decodable in the early latent steps, while s_5 becomes decodable only toward the end of the latent trajectory. Figure 13b shows an analogous effect for a 7-hop task, where s_6 appears earlier and s_7 appears later in the latent steps. This ordering is consistent with the analysis in Section 5: under composite moduli, the task becomes increasingly biased toward the last few updates, making late intermediates disproportionately predictive of the final answer s_{n+1} . CODI’s latent channel appears to exploit this structure by allocating its limited latent compute to tracking the last one or two intermediates rather than maintaining a full step-by-step rollout.

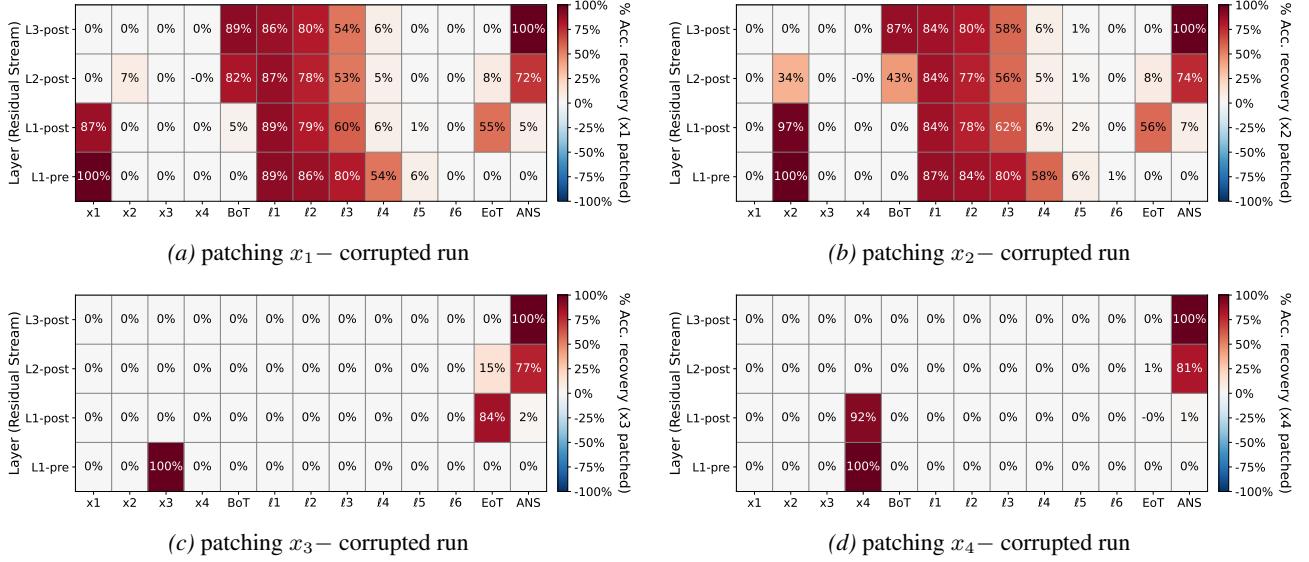


Figure 11. Activation patching for input-token corruptions in the three-hop polynomial task. When x_1 or x_2 is corrupted, patching clean activations into the latent-thought positions (ℓ_1, ℓ_2, ℓ_3) yields substantial accuracy recovery, indicating that the latent channel carries the intermediate information needed downstream (notably s_2 , which is required to compute s_3). When x_3 is corrupted, recovery concentrates at [EoT], suggesting that s_3 is computed or consolidated near the end of the latent segment. Finally, when x_4 is corrupted, recovery localizes at [Ans], consistent with a direct (copy-like) route that delivers x_4 to the answer readout.

H. Proofs for Theoretical Results in Section 5

H.1. Proof of Lemma 5.1:

Lemma 5.1 [Bijection criterion]: For $x \in R_m$, the map $f_x : R_m \rightarrow R_m$ is bijective iff x is a unit in R_m , i.e. $\gcd(x, m) = 1$.

Proof. The additive shift by b is a bijection, so f_x is bijective iff $s \mapsto sx$ is bijective. Multiplication by x is bijective over R_m iff there exists x^{-1} with $xx^{-1} \equiv 1 \pmod{m}$, which holds iff $\gcd(x, m) = 1$. \square

H.2. Proof of Lemma 5.2:

Lemma 5.2 [Exact contraction factor]: Let $m \geq 1$ and fix $x, b \in R_m := \mathbb{Z}/m\mathbb{Z}$. Let $d := \gcd(x, m)$. Define $\mu_x : R_m \rightarrow R_m$ by $\mu_x(s) = sx \pmod{m}$ and $f_x(s) = sx + b \pmod{m}$. Then $|\text{im}(\mu_x)| = m/d$, and every fiber of μ_x has size d . Equivalently, f_x is a d -to-1 map.

Proof. We view R_m as an additive group. The map μ_x is a group homomorphism since

$$\mu_x(s+t) = (s+t)x \equiv sx + tx \equiv \mu_x(s) + \mu_x(t) \pmod{m}.$$

Let $K := \ker(\mu_x) = \{s \in R_m : sx \equiv 0 \pmod{m}\}$. Write $x = dx'$ and $m = dm'$ with $\gcd(x', m') = 1$. Then

$$sx \equiv 0 \pmod{m} \iff dm' | s(dx') \iff m' | sx' \iff m' | s,$$

where the last equivalence uses $\gcd(x', m') = 1$ (so multiplication by x' is invertible modulo m'). Thus

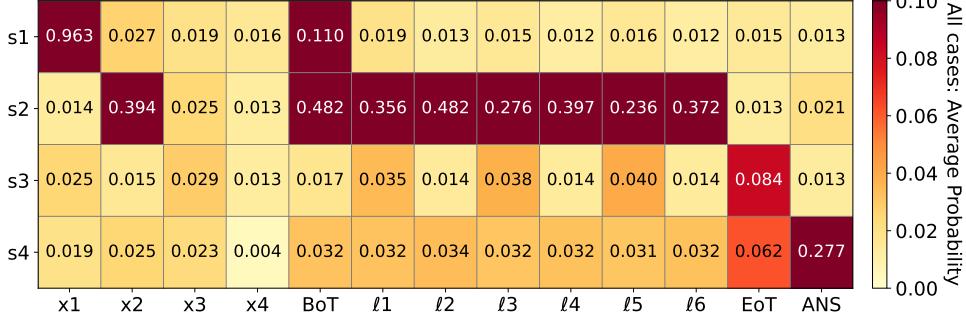
$$K = \{0, m', 2m', \dots, (d-1)m'\} \subset R_m,$$

so $|K| = d$.

For a group homomorphism $\varphi : G \rightarrow H$, each fiber $\varphi^{-1}(y)$ (for $y \in \text{im}(\varphi)$) is a coset of $\ker(\varphi)$: if $\varphi(s_0) = y$, then $\varphi^{-1}(y) = s_0 + \ker(\varphi)$. Hence every fiber has size $|\ker(\varphi)|$. Applying this to μ_x , every fiber has size $|K| = d$.

Since R_m is finite, the fibers partition R_m into $|\text{im}(\mu_x)|$ disjoint sets of equal size d , so

$$|\text{im}(\mu_x)| = \frac{|R_m|}{d} = \frac{m}{d}.$$



Ablation on the loss function Recall that CODI is trained with three losses: (i) a *teacher* objective that predicts an explicit CoT / state trace, (ii) a *student* objective that predicts the final answer after the latent-thought steps, and (iii) a *feature-space distillation* term that aligns teacher and student representations near the answer boundary.

Ablating distillation only. We first remove the distillation term while keeping the teacher and student losses. This ablation is still meaningful because the teacher and student share the same backbone and hyperparameters, so the teacher objective continues to shape the shared representation space that the student can leverage. Empirically, our main mechanistic signatures persist: the model still forms the late intermediate state s_n (and occasionally both s_{n-1} and s_n) in the final latent steps, and we still observe a specialized attention head that routes information from the final input token x_{n+1} to the [Ans] boundary. This suggests that the distillation term is not the primary driver of the partial (late-only) latent rollout in this setting.

Ablating both distillation and the teacher loss. Next, we remove both the distillation term and the teacher objective, leaving only the student loss. In this case, we no longer observe the late-step reasoning trace: the final latent steps do not reliably encode s_n (or (s_{n-1}, s_n)). This indicates that the teacher loss is important for inducing the late mechanism—likely because it pressures the shared backbone to represent stepwise state information, which the student objective can then compress into a short, late-bottleneck computation that still supports accurate answers.

J. Logit-Lens Visualization on Non-CoT Standard Transformer

Figure 14 shows the logit-lens visualizations for the standard (Non-CoT) Transformer.

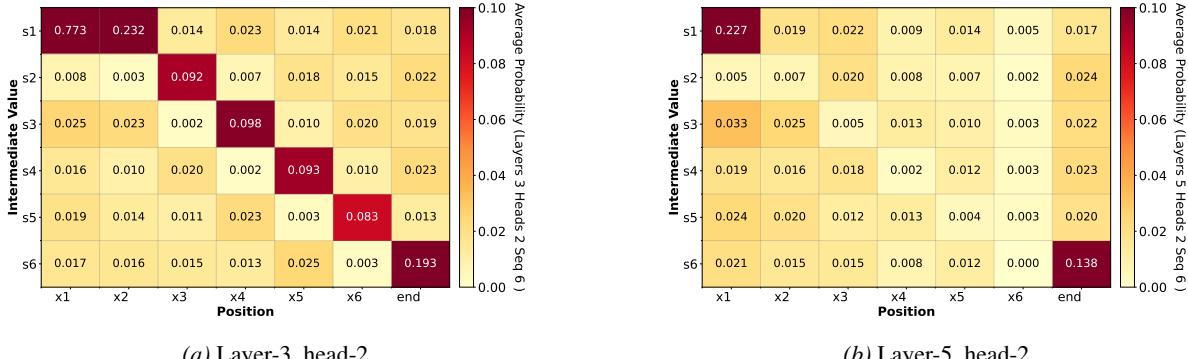


Figure 14. Logit-lens visualizations for the 5-hop task on all inputs. We compare a layer-3/head-2 model to a layer-5/head-2 model. The layer-3/head-2 model exhibits a clear rollout of intermediate states, whereas the layer-5/head-2 model shows little to no intermediate-state trace. This highlights the brittleness of step-by-step reasoning in standard (Non-CoT) Transformers and contrasts with latent-CoT models, which often converge to a late-bottleneck strategy.